TimeGAN Optimization on Small Datasets: Generating and Evaluating Synthetic Macroeconomic Time Series

Kincaid Fries
kfries@oxy.edu
Occidental College

1 Introduction

1.1 Background

Machine learning has been widely adopted in stock market prediction due to the abundance of stock data and the complicated, multivariable relationships that determine stock prices. In contrast, Baltazar, Reis, and Amorim (2020) [2] note that macroeconomic machine learning (ML) applications are much less reliable due to extensive surveying that takes months of research to produce even a single viable measurement. Additional challenges like varied calculation methods, unavailability of data for certain time periods, and changes in indicator definitions across countries make it hard to train macroeconomic models without overfitting to small datasets. I want to investigate synthetic data as an alternative avenue for allowing researchers to create realistic simulations of economic behavior without relying solely on historical data.

1.2 Goal

The goal of my senior comprehensive project is to evaluate whether a machine learning model trained on small macroeconomic datasets can be used to generate realistic synthetic time series. I hope to address the issue of relatively limited data availability per country by taking training data from multiple countries. My project will explore whether synthetic time-series could potentially address the problem of lack of data availability, and serve as a valid counterpart to real economic data in forecasting applications.

Data availability is an increasingly prevalent issue as the world embraces machine learning. The demand for training data is already outpacing availability, with major data companies even turning to piracy as a solution. If we had the power to take small datasets and extend them, or create realistic alternative data using synthetic generation, then these problems could be solved. Security, privacy, and copyright could be better preserved while still having access to the quantity of data needed to train machine learning models.

1.3 Approach

I will implement a TimeGAN model, which combines a supervised autoencoder to learn temporal dynamics with a generative adversarial network (GAN) to generate realistic synthetic data (Yoon, Jarrett, and Schaar (2019) [12]). The model will be trained using macroeconomic data taken from developed, free-market economies, to ensure there is consistency in the underlying macroeconomic trends it identifies. I will compare the properties of the synthetic data with real data using a number of statistical tests on distributional similarity, temporal structure, and novelty. I will examine the performance of a forecasting model using the train-synthetic test-real (TSTR) method to assess the potential viability of synthetic data in downstream ML training. Through consideration of both statistical factors and forecasting performance, I will develop a comprehensive framework for assessing the viability of the synthetic data. I will then refine the model, and explore optimization over limited training data.

2 Technical Background

2.1 Macroeconomic Indicators

Macroeconomic indicators are data points which quantify the health of an entire economic market at a country-wide level. They are impactful across all levels of society and business, and are even politically salient [8]. The macroeconomic indicators that I will focus on will be gross domestic product growth (GDP), inflation rate (I), unemployment rate (R), and population growth (P), which were selected with insight from the research of Ravallion (2021) [8] and Baltazar, Reis, and Amorim (2020) [2].

2.1.1 Selected Indicators

GDP: Total monetary value of all final goods and services in an economy. I will be using the rate of GDP growth expressed as a percentage change in GDP for this project.

Inflation Rate: Percent change in price of a pre-defined 'basket' of common household goods. Represents the change in currency value over time.

Unemployment Rate: Percent of the labor force who is unemployed and actively seeking work. The efficient unemployment rate is typically around 5% [4].

Population Growth: Percent change in the population. Though not typically considered a macroeconomic indicator, it helps control for other indicators. If GDP and unemployment rates both go up, a possible explanation would be an increase in population.

2.1.2 Relevance of Macro-Indicators

Macroeconomic indicators must be viewed in conjunction with each other, as no individual measure can capture the state of the economy [8]. Whenever one of the indicators changes, it will surely have an impact on the other indicators and the overall state of the economy. This project will look at each of the indicators outlined above in order to capture a holistic picture of the economy. Macroeconomic forecasting is primarily used by banks and the federal government. For example, Baltazar, Reis, and Amorim (2020) [2] detail how macroeconomic models can be used by banks to find their best response to economic shocks, potentially helping to handle situations like the 2008 financial crisis.

2.2 Relevent Machine Learning Terminology

Supervised learning uses labelled data to guide the models learning but is not well suited for time series [12].

Unsupervised learning identifies patterns without the guidance of labels.

Recurrant Neural Networks are artificial neural networks that pass data between neurons with a parameter allowing nodes to observe both the current and previous data state.

Generative Adversarial Networks (GANs) consist of a generator neural network competing with a discriminator [1]. The generator wants to beat the discriminator by producing data it can't tell is fake

Latent Space refers to the compression of input data into a lower dimensional space that preserves essential features of the data's structure [5].

Autoencoders learn to encode data to a latent space and then reconstruct it, allowing them to learn the structure of the data in the process [5].

2.3 Choosing TimeGAN

TimeGAN stands for Time-Series Generative Adversarial Network, a proprietary machine learning model created by Yoon, Jarrett, and Schaar (2019) [12] combining supervised, unsupervised, and adversarial learning to encourage the model to follow the temporal dynamics of training data. The goal of their research stems from the inadequacy of other machine learning models to generate synthetic data that accurately takes temporal correlations into account [12]. TimeGAN is trained on a 3D dataset of multiple independent time-series, shaped as $[N, L, D]$, where N is the number of sequences (e.g. countries), L is the number of time steps, and D is the number of dimensions (e.g., GDP, CPI). This makes TimeGAN perfect for my goal of training a model on multiple countries data.

2.4 TimeGAN Architecture

The main components of the TimeGAN architecture are pictured in Figure 1. Notably the supervisor is not pictured, but operates in the latent space between the embedder and recovery networks. $\delta\mathcal{L}$ represent the loss values, training parameters represented with θ.

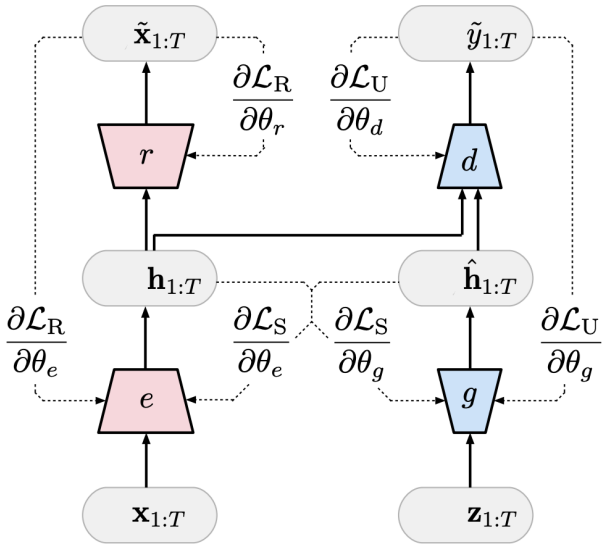

Figure 1: Components of TimeGAN Training Architecture. Adapted from Yoon, Jarrett, and Schaar (2019) [12].

Embedding Network (e, 1) encodes training data into a lower-dimensional latent space.

Recovery Network (r, 1) learns data relationships by decoding the data embedded in the latent space back into the original data-space. The combined embedding and recovery networks form an autoencoder in TimeGAN.

Supervisor Network (not pictured in diagram) works in the latent space learning the pattern between the next step hidden state by observing the previous. The super-

visor provides synthetic latent transitions to the Generator network, giving the Generator per-timestep guidance on how to generate data with accurate temporal transitions.

Generator Network (g, 1) generates sequences (time-series) from a random distribution of Gaussian noise vectors. Through competition with the discriminator and guidance from the supervisor, the generator learns how to convert random input into a realistic distribution.

Discriminator Network The discriminator network (d, 1) is the adversarial part of the model which attempts to distinguish between real and synthetic generations.

What makes this model unique is that the autoencoder (*e* and *r* in Figure 1) learns latent representations, while the supervisor learns time-step shifted latent transformations (the transition from latent representation at time *t* to time *t+1*). The latent patterns help the generator to create realistic data for the next time step based on the current time step. The adversarial component of the discriminator provides further reinforcement on generating quality synthetic sequences. The TimeGAN iteratively improves how it encodes features and generates representations during training [12]. The combination of GAN and autoencoder architecture make a TimeGAN the perfect model for my project: designed to generate realistic time series data.

2.5 Loss Functions

2.5.1 Autoencoder (reconstruction) Loss

The autoencoder uses mean squared error; a standard loss function in machine learning. The result is a quantification of the average difference between the input into the embedder and output from the recovery network, allowing the autoencoder portion of the model to learn better how to encode and recover data from the latent space.

$$\mathcal{L}_{\text{AE}} = \frac{1}{L-1} \sum_{t=0}^{L-1} \mathbb{E}\left[\left\| X_t - \hat{X}_t \right\|^2\right] \tag{1}$$

X_t is the real data fed into the embedder at time step t, and $\hat{X}_t$ is the data recovered from the latent embedding by the recovery network. The sum of squared error terms is averaged over the number of time steps L (the length of the series).

2.5.2 Supervised Loss

The supervised loss is also a mean squared error computed over latent representations shifted by one time step. Here, $\hat{H}_{t+1}$ denotes the supervisor network's prediction of the latent state at time $t+1$, and H_{t+1} denotes the true latent state at time $t+1$. The supervised loss encourages the model to learn realistic temporal transitions in the latent space.

$$\mathcal{L}_{\text{Sup}} = \frac{1}{L-1} \sum_{t=0}^{L-2} \mathbb{E}\left[\left\| H_{t+1} - \hat{H}_{t+1} \right\|^2\right] \tag{2}$$

2.5.3 Generator Loss

The generator loss has two components. First there is the adversarial loss, which is the component that comes from the discriminators ability to correctly identify the real data from the synthetic. Second is the supervised loss, which is weighted by a gamma coefficient that simply represents the amount of emphasis placed on supervised versus adversarial loss.

$$\mathcal{L}_{\text{G}} = -\mathbb{E}\left[\log\left(\sigma\left(D_{\text{logit}}\left(\hat{H}\right)\right)\right)\right] + \gamma \mathcal{L}_{\text{Sup}} \tag{3}$$

The adversarial loss uses logits, which are raw unbounded output of the discriminator on the synthetic latent sequence $\hat{H}$. Logits are passed through the sigmoid function which compresses it into a probability from $[0, 1]$. $\sigma\big(D_{\text{logit}}(\hat{H})\big)$ is the probability of seeing a logit value of $D_{\text{logit}}(\hat{H})$. The logit-sigmoid process internally makes the derivatives of the loss function easier to derive. The generator is trying to minimize the probability that its synthetic data is labelled synthetic by the generator, so the adversarial element of this loss function is negative.

2.5.4 Discriminator Loss

The discriminator loss function uses a standard function for yes no classification called binary cross-entropy. Binary cross-entropy is made up of two components, the probability score of assigning fake data the label of fake, and the probability score of assigning real data the label of fake.

$$\mathcal{L}_{\text{D}} = -\mathbb{E}\left[\log\left(\sigma(D_{\text{logit}}(H))\right)\right] - \mathbb{E}\left[\log\left(1 - \sigma\left(D_{\text{logit}}\left(\hat{H}\right)\right)\right)\right] \tag{4}$$

Both the supervisor and discriminator use expectation notation $\mathbb{E}[\cdot]$, which denotes the integral over the data probability distribution, e.g. $\mathbb{E}_{x\sim p(x)}[f(x)] = \int f(x)\, p(x)\, dx$, and in practice is approximated by a finite-sample average over a mini-batch: $\mathbb{E}_{x\sim p(x)}[f(x)] \approx \frac{1}{m}\sum_{i=1}^{m} f(x_i)$, where m is the mini-batch size and x_i are the samples.

3 Prior Work

The original TimeGAN implementation arose due to the fact that GANs "do not adequately attend to temporal correlations", while supervised models are "inherently deterministic" ([12], 1) . The proposal for addressing this disparity was to create an interwoven framework that provides

both the benefits of supervised models and the adversarial component of GANs [12] . My project is based on the TimeGAN model framework, which stood out to me because it is trained on batches of data that make it ideal for learning macroeconomic conditions across different countries. Additionally the combination of adversarial and supervised objectives offered the possibility to support learning on small datasets without overfitting.

My project draws primarily on four studies that reflect the technical, theoretical, and evaluative aspects of synthetic data generation and machine learning applications in economics. The first applies GANs to generating synthetic financial scenarios (Rizzato et al. (2023) [9]). The next applies a macroeconomic model to loan default rate predictions (Baltazar, Reis, and Amorim (2020) [2]). Finally, Yuan, Liu, and Cheng (2024) [13] and Livieris et al. (2024) [6] provide a reference for how to go about my evaluation metrics and results discussion.

3.1 Generative Adversarial Networks Applied to Synthetic Financial Scenarios Generation

Rizzato et al. (2023) [9] propose a GAN approach to synthetic financial scenarios called Jinkou. Jinkou was designed to generate synthetic time series datasets on the movement of equities under different macroeconomic conditions [9]. Unlike the original TimeGAN paper, they introduce a macroeconomic element which offers a similar conceptual approach and methodology to my own goals for this project.

3.1.1 Technical Approach

They took data on equities movements and augmented it with global state variables describing macroeconomic conditions [9]. Their proprietary Jinkou method uses a combination of a bidirectional GAN (BiGAN) and conditional GAN (cGAN) to improve the probability distribution of the synthetic data [9].

The BiGAN maps latent space to the data space with an autoencoder, similar to the embedder and recovery networks of the TimeGAN, and produces a synthetic dataset. The cGAN refines the results of the BiGAN by mapping each datapoint y to a probability score p drawn from the dataset distribution of P [9]. They experiment with four different market conditions (bull market, bear market, volatile market, and debt crisis) and evaluate the change in stock price from start to end of the time series for real vs synthetic data [9]. They found similar values for synthetic vs real data, typically with a difference between [-1,2]%, indicating that their synthetic data was accurate to real data predictions.

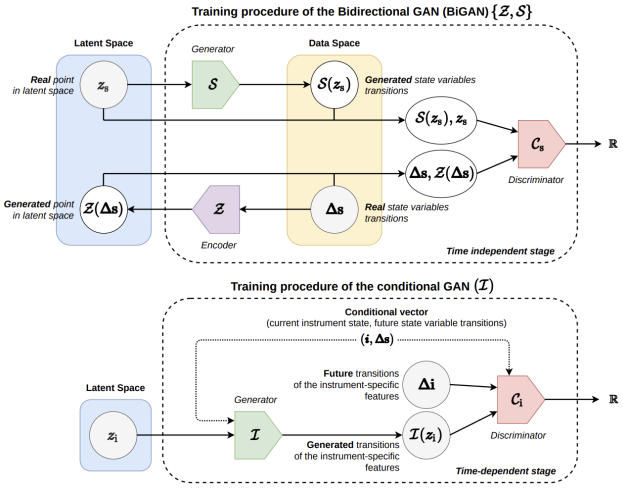

Figure 2: Visual comparison of BiGAN and Conditional GAN training architectures. Adapted from Rizzato et al. (2023) [9].

3.1.2 Key Takeaway

Specific to my project, Rizzato et al. (2023) [9] found the use of GAN objectives to be effective for generating synthetic time-series, and were able to compare that synthetic data to real data with a satisfactory degree of accuracy. Their results validate the importance of macroeconomic indicators in financial modelling.

3.2 Sustainable Economies: Using a Macroeconomic Model to Predict How the Default Rate is Affected Under Economic Stress Scenarios

Baltazar, Reis, and Amorim (2020) [2] build a macroeconomic stress-testing framework to predict loan default rates under hypothetical, simulated macroeconomic conditions. A framework they argue would allow banks and policy makers to improve their decision making in economic downturns Baltazar, Reis, and Amorim (2020) [2]. They use a predefined macroeconomic model rather than machine learning, having that model generate random Monte Carlo simulations Baltazar, Reis, and Amorim (2020) [2].

Baltazar, Reis, and Amorim (2020) [2] acknowledge the problem of small datasets for macroeconomic forecasting. Their study validates the use of GDP and inflation rate for understanding economic conditions, the downstream effects of macroeconomic conditions on finance and the economy, and the necessity and potential applications of synthetic macroeconomic data in augmenting existing datasets [2].

3.3 A Multi-Faceted Evaluation Framework for Assessing Synthetic Data

This study, by Yuan, Liu, and Cheng (2024) [13], proposes a novel method for the evaluation of synthetic datasets, which they call SynEval. Their evaluation method has three dimensions–fidelity, utility and privacy–of which I am interested in 'utility', their measure of the applicability of the synthetic data [13].

3.3.1 Technical Approach

Yuan, Liu, and Cheng (2024) [13] refer to the 'utility' of a synthetic dataset as its ability to be used downstream for machine learning. Downstream machine learning refers to the application of the generated data to training/testing machine learning models. The utility evaluation they use is Train-Synthetic-Test-Real (TSTR) [13]. They perform a study on synthetic data generated by popular LLMs (ChatGPT 3.5, Claude 3 Opus, and Llama 2 13B) [13]. They then trained four sentiment classification models; one on a real dataset, and the other three on the synthetic datasets produced by the LLMs [13]. The sentiment classification models were evaluated on unseen test data, with Mean Absolute Error (MAE) and percentage accuracy of correct classifications used to evaluate the performance of each model.

3.3.2 Key Takeaway

In evaluating the synthetic datasets on downstream machine learning performance, they found similar results between models trained on synthetic and real data, which demonstrates that synthetic data can be effectively used on downstream machine learning tasks [13]. Using downstream machine learning as an evaluation metric for the quality of synthetic data validates my plan of using a forecasting model trained on synthetic data as an evaluation metric.

3.4 An Evaluation Framework for Synthetic Data Generation Models

Livieris et al. (2024) [6] discuss a number of statistical methods specifically for evaluating synthetic data. They compare data generated by five synthetic data generation models using standard tests. The two tests they used which I find most relevant to my project were Wasserstein-Cramer's V-test and novelty test [6].

- *Wasserstein-Cramer's V Test* combines the Wasserstein distance (WD) with Cramer's V to account for the distributions across different variables [6]. The test evaluates if synthetic data matches the distribution of real data accounting for shape and variance, with low scores being better. Since Wasserstein-Cramer's V Test scores are not bounded to a range, it is somewhat arbitrary determining what is a good score or not.
- *Novelty Test*: measures the ratio of unmatched instances between the synthetic and real dataset, where an instance is considered match if $|s_i - r_i| < \alpha$, with s_i representing a synthetic datapoint, r_i representing a real point, and α being some predefined threshold [6].

3.4.1 Key Takeaway

I will use Wasserstein Distance as part of my distributional evaluation. Implementing the combination with Cramer's V-test is not a commonly accepted practice and makes the results a bit less interpretable, so I will focus just on Wasserstein Distance rather than the combined test. Additionally, I will implement the novelty test through a simple nearest neighbors Euclidian distance calculation.

4 Methods

4.1 Sourcing Macroeconomic Data

For macroeconomic training data I used the World Bank's World Development Indicators (WDI) online database [11]. I gathered a training dataset of the USA, Japan, Australia, UK, Canada, South Korea, Denmark, Germany and France–countries which I chose for having developed economies with reliable data going back 50+ years.

4.2 Data Pre-Processing

My data points of GDP growth, inflation rate, unemployment rate, and population growth were chosen deliberately to represent a snapshot of countries macroeconomic conditions without having collinearity between any of the features. GDP growth gives an idea of how the dollar quantification of the economy changes annually, while inflation rate helps control for the actual value of those dollars change adjusted for inflation. Unemployment rate along with inflation rate are considered two of the most revealing macroeconomic indicators, while population growth helps to control for some of the variation in unemployment and GDP growth.

Everything was processed using a standard min-max scalar compression to [-1,1], with the range of the data being saved in order to decompress the synthetic series at the end for comparison.

The most significant pre-processing decision I made was to divide the datasets into batches of sliding windows. With only 9 training countries I would have had a [9, 56, 4] training dataset (see 2.3 for training shape). This was simply too small of a dataset to do any real training on. As a way of combatting this, I broke each time series into a set of sliding

windows (see Figure 3). A window represents a sequence of a chosen length; in this case a 24 year period that is long enough to observe macroeconomic changes while allowing for the data to be divided up into many sequences. My stride length for the sliding windows was 1, so the start time of each window was incremented by 1. After dropping null values I was left with 155 windows available for training.

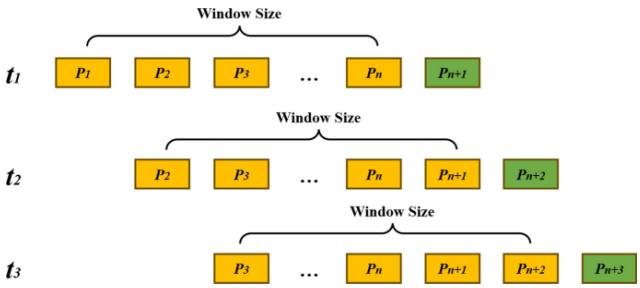

Figure 3: Visualization of sliding window approach to data batching. Colored boxes represent windows of length 4, with stride length 1. Adapted from Zhan and Kim (2024) [14].

Though windows will overlap, this method allows the network to learn the dynamics of short sequences through exposure to a much higher volume of sequences. With a [9, 56, 4] training dataset the network is exposed to 9x56=504 training steps. On a [64, 24, 4] training batch size as I used for many of my tests, the model is exposed to 64x24=1536 training steps. Even though there will be overlap, using 64/155 randomly sampled batches at a time minimizes overlap, and the increased training steps is worth the slight increase in risk of overfitting.

4.3 TimeGAN Implementation

For the actual TimeGAN model implementation itself I followed the original repository with some adaptation to my small dataset use case and different versioning requirements. The model is built on the TensorFlow library, with the data represented as tensors and the neural networks being built into as a tensor graph. Due to updates since the original TimeGAN paper, some structural changes were needed to switch to TensorFlow Keras versioning from the original. These changes affect the way some calls are made but do not affect the performance/implementation of the model.

For clarity's sake, I decided to move the actual neural cell instantiation into a helper function rather than directly call inside the neural networks as done in the original repository. This allowed me to treat RNN cell type as an adaptable parameter rather than a constant.

I used a Xaviar Initialization for the weight matrix of the neural connections. A Xaviar Init takes an input shape and outputs a random weight matrix with slight noise added for connecting the nodes within that shape. This introduces some slight variation and randomness into the weight matrix for each training iteration of the neural networks. With a small dataset the goal was to maximize justified, structured randomness to reduce overfitting.

I use an Adam Optimizer on the autoencoder, generator, and discriminator. The Adam Optimizer is an adaptive learning rate optimization algorithm (standard TensorFlow function) that adjusts the learning rate for features during training if it detects that certain features are being over or under emphasized.

4.4 Study Method

I will conduct evaluation on the synthetic generated data using the evaluation metrics 5, and iteratively adjust the model based on performance. My goal is to have a clear improvement between the performance of the stock model versus a final, tuned version, and be able to explain what changes I made and why they specifically are impactful to my small training set. Positive change could apply to improvement in the evaluation metrics or to an improvement in model health as indicated by loss values. Any parameter adjustments which have a notably positive impact I will treat as part of the new baseline, and will carry forward into the next trial.

After implementing the model and performing my iterative experiments over parameter adjustments, I hope to have a set of justified, meaningful changes that correlate to improved synthetic data quality. Through this approach I hope to justify that it is possible to optimize TimeGAN over small datasets to effectively generate synthetic data, as well as identify areas for future improvements in TimeGAN performance.

5 Evaluation Metrics

My goal is to statistically quantify the ability of the synthetic data to maintain distributional and temporal features while not overfitting training data. To achieve this I selected a number of evaluation metrics, as well as implemented a train-synthetic test-real approach as guided by prior work.

5.1 Feature Distribution: Kolmogorov-Smirnov

In order to evaluate the per-feature distribution of my synthetic data I will use the Kolmogorov-Smirnov (KS) test. The KS test is a cumulative frequency distribution (CFD) comparison test, used by Bourou et al. (2021) [3] to evaluate the performance of GANs in generating synthetic data. The KS test finds the maximum vertical distance between

two CFDs – in this case a real CFD and synthetic CFD (see Figure 4).

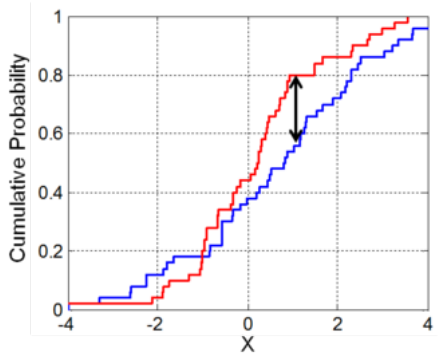

Figure 4: Visualization of Kolmogorov-Smirnov test between cumulative frequency distributions. From Wikipedia contributors (2025) [10].

KS uses the maximum distance calculation to perform a hypothesis test that the two distributions are not significantly different. If the p-value calculated by the test is less than your desired significance level α, then you can conclude that the sample distribution does not come from the original distribution. The KS test was run on each feature in the dataset, to give a sense of distributional similarity on a per-feature level.

5.2 Feature Distribution: Wasserstein Distance

The distribution test I will use is Wasserstein Distance, inspired by Livieris et al. (2024) [6] approach to synthetic data evaluation. WD measures the absolute difference (or cost) to transcribe one distribution onto another. Unlike the KS test which only looks at the individual point of highest difference, the WD looks at the total difference between CFDs. Like the KS test, I use a WD test on a per-feature basis to help evaluate whether one distribution came from another, in my case whether the synthetic distribution was derived from the real distribution.

Using WD and KS tests I will measure both the maximum and total cumulative difference between real-synth distributions. Since I know that the synthetic distributions came from the real, I will not be using these measurements for hypothesis testing reasons, but simply as a metric about the difference between distributions.

5.3 Cross Feature Correlation

I will use the Frobenius norm of correlation matrices to calculate cross feature correlation, inspired by Marti (2020) [7], who applies adversarial techniques to sampling on financial correlation matrices. A correlation matrix is a DxD matrix of features with each feature's correlation coefficient being stored in the corresponding box. In my case the correlation matrices would look like the following:

Table 1: Correlation Matrix of Macroeconomic Indicators

	GDP Growth	**Inflation (I)**	**...**
GDP Growth	1	$\text{corr}(\text{GDP}, \text{I})$	...
Inflation (I)	$\text{corr}(\text{I}, \text{GDP})$	1	...
...	...	...	...

A correlation matrix for both real vs real and for synthetic vs synthetic will be compared using the Frobenius norm, which calculates the total difference between matrices in the same conceptual manner as calculating Euclidean distance between vectors. Though there is not a standard to aim for in terms of Frobenius score, lower is better as it implies that the relationships between features are more similar. A lower Frob norm implies that the datasets have more realistic inter-feature relationships.

5.4 Autocorrelation

In order to more specifically consider temporal features, I included an autocorrelation function comparison (ACF). The ACF measures the correlation between a function and a time delayed version of itself. I compute the ACF score per feature for both the synthetic and real data. I then find the root mean squared error (RMSE) between both the average ACF score across features, and the per feature ACF score of synthetic and real datasets. Using RMSE ensures that larger differences are more heavily penalized. Since I am evaluating multiple features, even if most features have a good autocorrelation score it is alarming if one of them has a particularly bad score, which will be heavily weighted by the RMSE comparison. A lower average ACF RMSE indicates that the temporal features are more similar between the synthetic and real datasets.

5.5 Predictive (TSTR) Performance

As a test of downstream machine learning applicability, I used a train-synthetic test-real method of evaluation [13]. The premise is to evaluate whether synthetic data could be used as a viable proxy for real data by training a simple model purely on synthetic data, then evaluating its performance on real data. For my application, I used a stock linear regression model from TensorFlow trained on synthetic data to predict the next next timestep. The model's performance was then evaluated using the real data as a test batch, with the difference between actual and predicted quantified using mean squared error. The idea is that using real or synthetic data for training should be irrelevant as long as the synthetic data is realistic.

5.6 Novelty

The final essential dimension of synthetic data evaluation is novelty [6]. I use a K-Nearest Neighbors (KNN) test to find the nearest matched points between the synthetic and real data. First I normalize the distributions to 0 to ensure that different order-of-magnitude features do not skew the results. For every point in each distribution I find its nearest neighbor in terms of absolute Euclidean distance. I run this comparison from both synthetic to real and real to synthetic, which allows me to obtain a KNN asymmetry score (difference between synth-to-real and real-to-synth KNN mean) which tells me the direction the distribution is skewed toward. Ideally KNN scores would be high (indicating plenty of novelty) but KNN asymmetry would be low, meaning the new points are equally distributed. A high KNN score represents that the synthetic data is introducing novelty into the synthetic data it is generating, which if possible while still maintaining distributional/temporal dynamics is the exact goal of this model.

6 Results and Discussion

6.1 Baseline Model Evaluation

My first test of the complete model used only the set of default parameters listed in Table 4 (Appendix B). As a baseline I ran the evaluation metrics over comparisons between real and synthetic data, and between real training and withheld data. This established a benchmark for the evaluation methods.

Throughout testing the only evaluation metric which wasn't informative was the Kolmogorov–Smirnov (KS) test. Even the real–real comparison in Table 2 reports a KS Max of 0.875, which corresponds to an effectively zero p-value. I attribute this to differing starting conditions across countries and periods. Windows that include economic shocks for example would have significantly different CDFs, causing high KS distances even between real samples. Since two distributions can be real and have significantly different underlying macroeconomic scenarios, KS is of limited use here.

Overall my synthetic baseline (v0) test performs similarly to the real-real comparison except for on realism and predictive scores. Both WD mean and max are higher in v0, but are relatively similar (Table 2). This indicates that the model is getting rough distributional shapes, but is performing poorly at more extreme values (higher WD Max). Thus marginal distributions are not well matched, but distributional closeness is not drastically different from the real baseline.

There was a large disparity in performance in the Frobenius difference in correlation matrices. The real baseline had a Frob score of 1.438, while v0 had a significantly higher score of 3.432 (Table 2). This indicates that the v0 synthetic data does not at all mimic the inter-feature relationships of the real data well. Surprisingly v0 has slightly better ACF RMSE than the real-real baseline. Most likely this is due to the fact that early GAN outputs could be oversmoothed, generating series without much temporal variation that creates artificially low autocorrelation, which would explain some of the poor performance in Frobenius difference.

Table 2: Comparison of evaluation metrics between the real–real baseline and the initial synthetic baseline (v0).

Metric	Real Baseline	v0 Synthetic
KS Max	0.875	0.975
WD Mean	0.174	0.206
WD Max	0.341	0.384
Frob Diff	1.438	3.432
ACF RMSE	0.174	0.155
TSTR MSE Mean	0.037	1.096
KNN Mean S–R	14.361	12.728
KNN Mean R–S	13.725	15.755
KNN Asymmetry	0.318	-3.144

The most important difference between the two tests is the TSTR performance. The TSTR MSE of the v0 test was 1.096, significantly worse than the real baseline of 0.037 (Table 2). The poor predictive score indicates that the synthetic data cannot reasonably be used to predict real data relationships. The synthetic data is not realistic enough for any sort of downstream application, which is how Yuan, Liu, and Cheng (2024) [13] defines the real-world utility of synthetic data.

In terms of novelty it seems however that a score of about 14 is realistic given the results of the real-real comparison. What stands out is that for the v0 test, there was a much higher degree of asymmetry (-3.144 vs 0.318) between the real-fake and fake-real KNN means (Table 2). This implies that the synthetic data is asymmetric, meaning that v0 synthetic samples cluster in certain ranges (lower synthetic to real score) but fail to cover the full range of real data (higher real to synthetic score).

6.2 Final Results: Tuned Model

From my initial baseline to my final trial (v14) there was substantial improvement across distributional, temporal, discriminative, and most importantly downstream application performance. See Appendix C.1 for further discussion of intermediate results and trials.

The final incremental change from v10 to v14 was increasing hidden dimensions from 24-32. The hidden dimensions directly control the capacity of the embedder, super-

Table 3: Performance improvements from the initial model (v0) to the tuned model (v14).

Metric	v0	v14	Improvement
KS Max	0.975	0.975	0%
WD Mean	0.206	0.183	12.56%
WD Max	0.384	0.362	6.07%
Frob Diff	3.432	2.621	30.94%
ACF RMSE	0.155	0.287	–45.99%
TSTR Mean	1.096	0.588	86.39%
KNN Mean S–R	12.728	13.202	–
KNN Mean R–S	15.755	15.611	–
KNN Asymmetry	-3.144	-2.409	30.51%

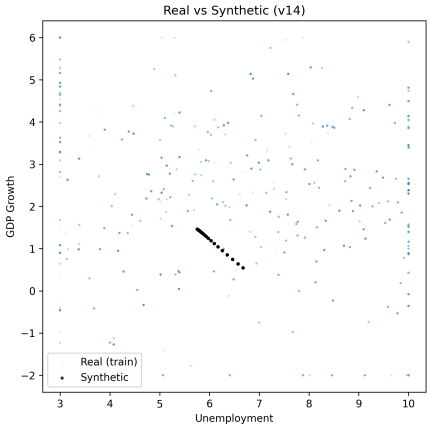

Figure 5: GDP Growth vs Unemployment Rate for v14 Synthetic vs Real Data

visor, generator, and recovery networks exposure to previous hidden states. The increased hidden dimensions better allow the supervisor to model lagged relationships, which helps inform how the generator generates realistic synthetic time steps.

Compared to v0, WD mean and max showed a moderate improvement, of 12.56% and 6.07% respectively–indicating that the model matches the shape of the real distributions better, and that distributions are not as narrowly clustered as in v0 (Table 3). Major improvement was seen in Frobenius difference, a 30.94% improvement over v0, indicating that feature to feature relationships were far more accurate than initially (3). This implies that the synthetic data has much more realistic features to feature relationships than the baseline.

The only metric where v14 actually performed worse than v0 was the autocorrelation structure (ACF RMSE). The change from an ACF RMSE of 0.155 in v0 to 0.287 in v14 indicates that there was more temporal volatility in v14 (Table 3). While this is a significantly worse score, in terms of my goals it is not necessarily concerning. The initial ACF RMSE score for v0 was lower than the real-real baseline (Table 3). This implies that the initial ACF RMSE measurements over-smoothed temporal dynamics, when in reality they may be more erratic.

The most impactful improvement from v0 to v14 is the decrease in TSTR Mean from 1.096 in v0 to 0.588 in v14 (Table 3). The v14 trial was 86.39% more accurate than v0 when it came to using synthetic data to predict the next real data step. With further refinement it may be possible to reach a point where synthetic data serves as a valid proxy for real data for downstream machine learning applications.

The v14 test had the lowest KNN asymmetry of all tests of -2.409, though still far from the real-real baseline (Table 3). As asymmetry moves closer to zero that means that the synthetic data more uniformly covers the real data distribution and shows less of a tendency to cluster together.

Visually the v14 distribution is clustered around the mean, as it is easy for a model with exposure to a small amount of data to minimize average loss by just generating average values (see 5). However, throughout testing I found that with the right adjustments the model can be tuned to introduce novelty outside of average values without sacrificing realism. In terms of empirical applications, until a TSTR score comparable to the real-real baseline (see 2) is reached the synthetic data cannot be viably used downstream.

Overall the improvements in evaluation metrics demonstrate that adversarial techniques could viably produce synthetic data from small training sets. The combined supervised and adversarial objectives of TimeGAN show the potential of the model to capture the subtle temporal relationships between macroeconomic indicators.

6.3 Avenue for Future Study

I observed that adversarial learning may too heavily penalize generations that are different from the mean when trained on such small datasets. Generating synthetic data at the mean of a distribution is the easiest way to produce stable, low average loss. The efforts that I made to decrease the influence of the adversarial component of the model, such as adding noise to the discriminator and increasing gamma lead to some of the most significant improvements in model performance. Therefore in future research I would like to compare the performance of TimeGAN on small datasets to non-adversarial models.

For example, a framework like variational autoencoders (VAEs) could be applied to synthetic time series generation. VAEs are probabilistic models which learn a latent probability distribution and could then use that to generate a synthetic latent distribution which is then decoded into a synthetic dataset. It is somewhat the opposite of TimeGAN, which takes in a random distribution and attempts to gener-

ate it into a realistic synthetic distribution.

I would also like to apply my research to generating data from initial conditions. Macroeconomic data and decision making is based around the expected future movements of macroeconomic indicators. In my case, random macroeconomic datasets may have better downstream applications than a specific prediction based on initial conditions, but overall are less likely to be applicable to actual decision making. My biggest concern is that you would need substantial training data to learn patterns from each different initial condition.

7 Ethical Considerations

In a hypothetical scenario where synthetic macroeconomic data was to become widely used by economists/bankers, the primary ethical concern with my project becomes the question of who actually benefits from improved economic simulation, and who may be unintentionally excluded from those benefits.

Ravallion (2021) [8] points out how changes in macroeconomic conditions have disproportionately negative impacts on the poorest Americans. Unemployment or inflation rates for example, may negatively impact the *value* of the wealth of upper class Americans, but won't actually impact their livelihood [8]. The poor however, are placed under immense burden from macroeconomic changes.

The major concern is that improved macroeconomic modelling only serves to benefit those with the financial literacy and wealth to take advantage of it–particularly banks, policymakers, and wealth investors. As such, macroeconomic modelling is likely to be used to benefit these upper-class groups, rather than improve the financial standing and financial literacy of the lower class.

In a hypothetical world where synthetic data is widely adopted there is the risk that synthetic data is used by a powerful actor without disclosure of where that data came from. Since macroeconomic factors are impactful to all, anybody who is in a policy making position must be transparent and accountable for the data they use. The general public, and even data-scientists, may not know exactly what patterns an algorithm is learning in training. There is the potential that synthetic data abstracts real world conditions, creating incomplete or untrustworthy datasets that are then used in decision making.

The use of algorithmic decision making in credit-scoring and loan rating is controversial, as it obscures the process behind making very impactful decisions on people's lives. The use of synthetic macroeconomic data for decision making poses the same risks–obscuring factors in decisions which impact people's lives. If any bias is present in the dataset that is used to produce the synthetic data, then the bias patterns will be present in the synthetic data as well. When it comes to real people's lives, using synthetic data becomes a real liability.

8 Conclusion and Future Work

I was able to make notable improvements in the performance of the TimeGAN model on generating synthetic data from a small input dataset. Both the statistical and TSTR scores improved from my v0 to v14 tests. However, I was unable to fully prevent overfitting to the input data and the tendency to collapse toward the mean. There is an inherent conflict between generating novel, varied distributions, and loss functions which aim to minimize difference. I was able to refine this tradeoff to an extent, and produce synthetic data in v14 which demonstrated both better novelty, predictive, and distributional scores than in v0. Ultimately though I would conclude that in its current state, the size of the training dataset is too severe a limiting factor in the ability to produce temporally realistic and distributionally viable synthetic datasets using TimeGAN.

If a model were to reach the goal of producing realistic synthetic macroeconomic data to the extent that said synthetic data was indistinguishable from real, then this raises a number of ethical concerns. Aside from the obvious like fabricating data, if applied to macroeconomic modelling techniques there is a risk of further exacerbating the wealth divide or people being harmed by bias synthetic data produced from bias training data. Transparency on the generation process of synthetic data is vital to mitigating these risks.

References

[1] Amazon Web Services. *What is a Generative Adversarial Network (GAN)?* https://aws.amazon.com/what-is/gan/. Accessed: 2024-04-28. 2023.

[2] Baltazar, José, Reis, João, and Amorim, Marlene. "Sustainable economies: Using a macro-economic model to predict how the default rate is affected under economic stress scenarios". In: *Sustainable Futures* 2 (2020). ISSN: 2666-1888. DOI: https://doi.org/10.1016/j.sftr.2020.100011.

[3] Bourou, Stavroula et al. "A Review of Tabular Data Synthesis Using GANs on an IDS Dataset". In: *Information* 12.9 (2021). ISSN: 2078-2489. DOI: 10.3390/info12090375. URL: https://www.mdpi.com/2078-2489/12/9/375.

[4] Cazes, Sandrine and Verick, Sher. *Figuring Out Efficient Unemployment*. https://cepr.org/voxeu/columns/figuring-out-efficient-unemployment. Accessed: 2024-04-28. 2020.

[5] IBM Research. *What is Latent Space?* https://www.ibm.com/think/topics/latent-space. Accessed: 2024-04-28. 2023.

[6] Livieris, I. E. et al. "An Evaluation Framework for Synthetic Data Generation Models". In: *Artificial Intelligence Applications and Innovations*. Ed. by Maglogiannis, Ilias et al. Cham: Springer Nature Switzerland, 2024, pp. 320–335. ISBN: 978-3-031-63219-8.

[7] Marti, Gautier. "CORRGAN: Sampling Realistic Financial Correlation Matrices Using Generative Adversarial Networks". In: *2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. arXiv preprint arXiv:1910.09504 (v2, Dec 2019). IEEE, 2020, pp. 8459–8463. DOI: 10.1109/ICASSP40776.2020.9054450.

[8] Ravallion, Martin. *Macroeconomic Misery by Levels of Income in America*. Working Paper 29050. National Bureau of Economic Research, July 2021. DOI: 10.3386/w29050. URL: http://www.nber.org/papers/w29050.

[9] Rizzato, Matteo et al. "Generative Adversarial Networks applied to synthetic financial scenarios generation". In: *Physica A: Statistical Mechanics and its Applications* 623 (2023), p. 128899. ISSN: 0378-4371. DOI: https://doi.org/10.1016/j.physa.2023.128899.

[10] Wikipedia contributors. *Kolmogorov–Smirnov test*. https://en.wikipedia.org/wiki/Kolmogorov-Smirnov_test. Accessed: 2025-12-05. 2025.

[11] World Bank. *World Development Indicators*. https://databank.worldbank.org/source/world-development-indicators. Data retrieved for the period 1970–2024. 2024.

[12] Yoon, Jinsung, Jarrett, Daniel, and Schaar, Mihaela van der. "Time-series Generative Adversarial Networks". In: *Advances in Neural Information Processing Systems*. Ed. by Wallach, H. et al. Vol. 32. Curran Associates, Inc., 2019. URL: https://proceedings.neurips.cc/paper_files/paper/2019/file/c9efe5f26cd17ba6216bbe2a7d26d490-Paper.pdf.

[13] Yuan, Yefeng, Liu, Yuhong, and Cheng, Liang. *A Multi-Faceted Evaluation Framework for Assessing Synthetic Data Generated by Large Language Models*. 2024. arXiv: 2404.14445 [cs.LG]. URL: https://arxiv.org/abs/2404.14445.

[14] Zhan, Zeqiye and Kim, Song-Kyoo. "Versatile time-window sliding machine learning techniques for stock market forecasting". In: *Artificial Intelligence Review* 57.8 (2024), p. 209. DOI: 10.1007/s10462-024-10851-x.

A Replication Instructions

A.1 Software and Versioning

The codebase for this project was built in using the VSCode IDE, with all programming aspects of the program done in Python version 3.13.3. The programs were run on a Mac with an M2 Pro chip and 16G RAM. The global data is stored in csv files in the data and datag2 folders. Notably in TimeGAN, TensorFlow uses the newest Keras V3 version rather than the older version used in the original TimeGAN implementation. The difference is that Keras V3 has different session management, so you have to ensure to disable eager execution in order to run the program in the same manner as the original (use tf.compat.v1.disable_eager_execution() at the top of the file). With the Keras V3 version the sessions are instantiated and run slightly differently, and if there was a future update to TensorFlow versioning then you would need to either explicitly set the version back to Keras V3 or adapt to the new version.

As far as evaluation metrics and mathematical components are concerned, these all use numpy, pandas and SciKit libraries which are more static and less frequently changed than TensorFlow versioning. Still in future implementation pay attention to where these libraries are called as they may have deprecated features that need to be adapted. All libraries and versioning requirements are stored in the requirements.txt file in the repository

A.2 Run Instructions

1. **Prepare and clean the dataset.**
   - Ensure the raw macroeconomic dataset is correctly structured before passing it from `main.py` into `timegan.py`.
   - Although the initial goal was universal dataset compatibility, additional datasets require re-implementing:
     - `data_clean.py`
     - `prep_windows.py`

     to match file naming conventions and dataset dimensions.
   - The provided macroeconomic dataset works without modification.
   - The `g2` cleaning and window scripts serve as examples for adapting the pipeline to alternative datasets.
2. **Clean raw CSV into per-country files)**
   - File: `data_clean.py`
   - Arguments:
     - `--input`: Path to the raw dataset (default: `data/MacroData.csv`)
     - `--outdir`: Output folder for cleaned `Country*.csv` files (default: `data/clean`)
   - Example Usage:

```
# Run with defaults
python3 data_clean.py


# Specify a different input file
python3 data_clean.py --input data/alt/MacroData_alt.csv


# Specify a different output directory
python3 data_clean.py --outdir data/clean_custom


# Specify both
python3 data_clean.py --input raw/newdata.csv --outdir data/newclean
```

3. **Create Sliding Windows**
   - File: `prep_windows.py`
   - Arguments:

  - `--data_dir`: Directory containing cleaned `Country*.csv` files (default: `data/clean`)
  - `--L`: Sliding window length (default: 24)
  - `--stride`: Step size between windows (default: 1)
  - `--val_countries`: Validation countries (default: `Country7`)
  - `--test_countries`: Test countries (default: `Country8 Country9`)
- Example Usage:

```
# Run with defaults
python3 prep_windows.py


# Change window length
python3 prep_windows.py --L 36


# Specify custom countries
python3 prep_windows.py --val_countries Country5 Country6 --test_countries
Country7 Country8


# Use a different cleaned dataset folder
python3 prep_windows.py --data_dir data/clean_g2


# Full custom example
python3 prep_windows.py
--data_dir data/clean_g2
--L 36
--stride 2
--val_countries Brazil
--test_countries India China
```

4. **Run a TimeGAN Model Version**
   - File: `main.py`
   - Run a particular experiment version with:
     - `python3 main.py X`
     - `X` is the version identifier (e.g., `0`, `3`, `14`)
   - Versions are implemented as `elif` blocks in `main.py` to prevent overwriting results
   - Each version defines its own hyperparameter overrides
   - To create a new version:
     - Create a unique version label
     - Add a corresponding `elif` block in `main.py`
     - Call it through the command line as shown above
   - Using a new dataset requires adapting:
     - filename paths
     - input shapes
     - windowing behavior
     - dataset-specific normalization rules
5. **Synthetic Data Output**
   - Each run automatically creates: `artifacts/baseline_vX`
   - Synthetic sequences are saved as:
     - `synthetic_scaled.npy`
     - `synthetic_orig.npy`
   - Training, validation and test sets are also saved to the same folder for reference

6. **Run Evaluation Metrics**
    - File: `eval_metrics.py`
    - Run with: `python3 eval_metrics.py X`
    - Results saved as a `json` to: `artifacts/baseline_vX/results/results_vX_metrics.json`
7. **Generate Diagnostic Graphs**
    - File: `graphing.py`
    - Run with: `python3 graphing.py X`
    - Results saved as a `png` to: `artifacts/baseline_vX/`
    - By default, graphs are produced for GDP growth and unemployment rate
    - Modify `graphing.py` if using datasets with different indicator names

B Architecture Overview

B.1 High-Level Repository Layout

- **data/**
    - **clean/**
        - Per-country CSV files from the original 9-country dataset
        - `MacroData.csv` (raw input data from World Bank WDI)
    - **datag2/** (experimental; not essential)
        - **clean_g2/**
            - Cleaned data for all countries globally (experimental)
            - `MacroData_g2.csv`
- **artifacts/**
    - **baseline_vX/**
        - `config.json`
        - `train_scaled.npy`
        - `val_scaled.npy`
        - `test_scaled.npy`
        - `train_orig.npy`
        - `val_orig.npy`
        - `test_orig.npy`
        - `synthetic_scaled.npy`
        - `synthetic_orig.npy`
        - **results/**
            - `results_vX_metrics.json`
- `data_clean.py`
- `prep_windows.py`
- `utils.py`
- `timegan.py`
- `main.py`
- `eval_metrics.py`
- `graphing.py`
- `sanity.py`
- **.venv/** (Python virtual environment)

B.2 Layout Explanation

The original TimeGAN repository was less modular than my version. I decided that for my use case I wanted to break each file out into a clear and distinct purpose, which allows the program to run in a modular way, so if you want to adapt to new

data uses you can do so file by file. The artifacts folder contains all data results, with both the training/synthetic data saved in each file as well as the evaluation metrics and optionally graphics if you choose to run graphing.py on the trial number. Everything is based around trial numbers, and in main I chose to store trial numbers as elif statements because initially I was running into issues where I would change parameters and then accidentally overwrite results with different parameter runs. This way even if the model is re-run on the same trial number, it will use the original parameters used by that trial number and not rewrite old data. I do not think it is the most efficient and sustainable code design, but it has successfully gotten rid of the issue of rewriting trials.

B.3 File Descriptions

- **Data Cleaning: `data_clean.py`**
  - **Purpose:** Takes a raw macroeconomic CSV file (e.g., `MacroData.csv`) and outputs a set of per-country cleaned CSV files in `data/clean/`. This prepares the dataset for sliding-window generation and later model training.
  - **Key Uses:**
    - Load the raw CSV into a Pandas dataframe.
    - Normalize, pivot, and restructure columns (e.g., indicator/year formatting).
    - Remove NaN values and ensure consistent ordering across years.
    - Split the full dataset into individual `CountryX.csv` files.
    - `clean_datag2()`: performs the same process on the larger G2 dataset with different column naming conventions (experimental extension).
- **Window Preparation: `prep_windows.py`**
  - **Purpose:** Takes cleaned per-country CSV files and outputs NumPy arrays of sliding windows, stored as `*.npy` files in each `baseline_vX/` folder. This produces the `[N, L, D]` training arrays used by TimeGAN.
  - **Key Uses:**
    - Load cleaned country CSVs into Pandas dataframes.
    - Convert each country's time series into overlapping windows of length `L`.
    - Scale features to the range `[-1, 1]` to standardize magnitude.
    - Output summary dictionaries and NumPy arrays containing training, validation, and test window sets.
    - `prepare_windows_global()`: global variant used for the G2 dataset (not part of core experiment).
- **Utilities: `utils.py`**
  - **Purpose:** Provides helper functions and default model parameters. Takes simple Python objects (dicts, lists, scalars) and returns initialized TensorFlow components, RNN cells, optimizers, and parameter dictionaries.
  - **Key Uses:**
    - Set NumPy and TensorFlow seeds for reproducibility.
    - Implement Xavier initialization and Adam optimizer setup.
    - Store the base model configuration dictionary.
    - Construct RNN cells (either `lstm` or `gru`) and stack layers.
    - Generate utility functions used throughout model construction and training.
- **TimeGAN Model Implementation: `timegan.py`**
  - **Purpose:** Takes preprocessed training windows (`NumPy` arrays) and training parameters (dict), and outputs trained TensorFlow models and synthetic window samples. Also stores the computation graph and handles for running networks.
  - **Key Uses:**
    - Define Embedder, Recovery, Generator, Supervisor, and Discriminator networks.
    - Build and store TensorFlow computation graph components.
    - Implement all loss functions (AE, supervised, adversarial, moment losses).
    - Perform learning using the Adam optimizer.
    - Return a handles dictionary used to execute forward passes during synthetic data generation.
- **Training Orchestration: `main.py`**

  - **Purpose:** Takes cleaned window arrays and version-specific hyperparameter overrides, executes AE and GAN training loops, and outputs real and synthetic NumPy arrays into each `baseline_vX/` directory.
  - **Key Uses:**
    * Load and assemble windowed data from `prep_windows.py`.
    * Extract array shapes and initialize the full TimeGAN architecture.
    * Run the Autoencoder warmup stage then GAN training stages.
    * Save `train_orig.npy`, `test_orig.npy`, `synthetic_orig.npy`, and their scaled counterparts.

- **Evaluation Metrics: `eval_metrics.py`**
  - **Purpose:** Takes real and synthetic NumPy arrays from a `baseline_vX/` directory and outputs a JSON file of metric results.
  - **Key Uses:**
    * Load `*.npy` arrays for real and synthetic data.
    * Compute all evaluation metrics (KS Max, WD Mean/Max, Frobenius norm, ACF RMSE, TSTR, KNN symmetry metrics).
    * Store results in `results_vX_metrics.json`.

- **Visualization: `graphing.py`**
  - **Purpose:** Takes synthetic and real NumPy arrays for a version `X` and outputs PNG visualizations in the corresponding `baseline_vX/` folder.
  - **Key Uses:**
    * Load `train_orig.npy` and `synthetic_orig.npy`.
    * Plot GDP vs. Unemployment scatter distributions.
    * Plot GDP and Unemployment time series.
    * Save plots as PNG files for qualitative inspection.

- **Sanity Checks: `sanity.py`**
  - **Purpose:** A small diagnostic script used to ensure check accidental overwriting. Takes no input other than existing version folders and prints debugging information; not required for standard usage.

B.4 Default Training Parameters

Table 4: Default training parameters used in the baseline TimeGAN implementation.

Model Parameter	Default Value
L (sequence length)	24
D (feature dimension)	4
z_{dim}	4
batch_size	64
ae_warmup_it	600
gan_iters	3000
γ	1.0
learning_rate	0.001
module	`"gru"`
hidden_dim	24
num_layers	2

These are the training parameters passed into the model during training. They are also the primary hyperparameters varied during experimentation.

- L: Length of each sliding window (number of time steps). In this project, $L = 24$ corresponds to a 24-year sequence.
- D: Number of dimensions in the training data (number of economic indicators). This was fixed at $D = 4$ for GDP growth, inflation rate, unemployment rate, and population growth.

- z_dim: Dimensionality of the random noise vectors fed into the generator, controlling generative diversity.
- **batch_size**: Number of sliding windows passed to the model during each training step.
- *ae_warmup_it*: Number of autoencoder warmup iterations prior to adversarial training.
- *gan_iters*: Number of full GAN training iterations.
- γ: Weighting coefficient balancing the supervised loss and adversarial loss components within the generator loss.
- **learning_rate**: The step size used during optimization; controls how quickly model weights update during backpropagation.
- *module*: The type of recurrent cell used in the model's networks. Supported options include:
  - ``gru'': Gated Recurrent Unit; uses an update gate (memory retention) and reset gate (context forgetting). More lightweight and computationally efficient.
  - ``lstm'': Long Short-Term Memory; uses three gates and maintains a cell state, offering greater robustness at higher computational cost.
- *hidden_dim*: Size of the hidden state in each neural network layer; determines the number of latent features used to encode temporal structure.
- *num_layers*: Number of stacked layers in the recurrent neural network architecture.

C Additional Results and Exploration

C.1 Significant Trials and Results

In each different trial iteration I changed a different parameter of the model, and in some cases I made slight changes to the training process such as adding noise to the discriminator, or changing RNN cell type. Some of the most notable version changes are shown in the table below. The bolded cells represent the best performance of that evaluation metric across the sample experiments.

Table 5: Intermediate model evaluation metrics across selected development versions (v0, v3, v4, v8, v10).

Metric	v0	v3	v4	v8	v10
KS Max	0.975	0.969	0.951	0.976	0.975
WD Mean	0.206	0.189	0.176	0.214	0.181
WD Max	0.384	0.319	0.289	0.377	0.346
Frob Diff	3.432	2.304	3.479	2.969	2.909
ACF RMSE	0.155	0.278	0.132	0.147	0.225
TSTR Mean	1.096	3.554	1.559	1.246	0.251
KNN Mean S–R	12.728	12.302	11.082	13.209	12.328
KNN Mean R–S	15.755	15.929	15.955	16.291	15.486
KNN Asymmetry	-3.144	-3.627	-4.873	-3.018	-3.157

In v3 I increased the iterations of the autoencoder warmup from 600 to 1000. The goal was to allow the model more time to learn latent encoding and transformations before the adversarial component was trained. This effect showed very clearly in the Frobenius difference, with a significant improvement in cross-feature correlations. Extending the autoencoder warmup iterations also showed improvements from v0 in distributional metrics (WD) (5). Although this came at the expense of ACF and TSTR performance, for early testing I thought that the improvement in correlation matrices was worth keeping this change in as a point to further build from.

In v4 I decreased the learning rate from 0.001 to 0.0001, which led to substantial improvement in distributional similarity and the best autocorrelation score across all tests (5). The goal was to reduce overfitting on the small dataset by decreasing the rate at which the model learns. The TSTR score was significantly lower from v3 to v4, which indicates that the actual downstream applicability of this version was far better than the previous, and back toward the baseline. In these early tests everything came almost as a direct tradeoff. In the areas where v4 showed improvement from v3, it also lost some of the improvements that v3 made over v0. Notably though, v4 had much smoother loss gradients in the supervised and autoencoder loss, which was a large part of the reason I was inclined to keep it in. The early iterations were largely about improving model health, observed primarily from loss values.

Between v4 and v8 progress stagnated, but in v8 I had somewhat of a breakthrough by adding Gaussian noise with a standard distribution of 0.02 to the input latent sequences. This served to add some additional randomness to the discriminator; a standard stabilization technique in GANs. It helps to prevent the discriminator from becoming too confident too early and slows down its convergence to give time for the generator to learn. With no noise, the discriminator quickly overfits to small differences in the latent space, but the added uncertainty in the discriminator loss helps the generator to better learn more erratic latent transitions. The result was improvements from v0 in Frob Diff, ACF RMSE, and KNN asymmetry, indicating that the model better represented intra-feature dependence, temporal patterns, and better filled the real distribution.

Interpreting loss values is tricky, because you do want them to converge toward 0 throughout training, but if they ever reach 0 then it is evidence of overfitting, and if they converge too low too quickly then it is evidence of collapse. With the autoencoder, lower loss is not necessarily correlated with better generations. For v10 I halved autoencoder loss in the actual model implementation. This helped decrease the extent that the autoencoder constrained latent space and gave the generator more freedom to generate synthetic data, as until this point all results tended to be repetitive and low-variance. Keeping autoencoder iterations high from v3 allows for convergence to stable loss values, while halving the loss ensures less pull toward exact reconstruction. The result of v10 was a test that was across the board similar to or better than all metrics in v0, but with a significant improvement in one of the most important metrics: TSTR mean. With a v10 score of 0.251, and every other iteration having a TSTR score of over 1, this was a clear and very significant improvement.

C.2 Additional Meaningful Changes

There were a few other changes that did not lead to notable changes in evaluative performance, but were observed to improve loss values into a more healthy range. The first was reducing GAN iterations from 3000 to 2000, a change done in v1 that was kept through to the final version v14. In v0, the loss values plummeted (see 6), indicating clear collapse of loss values and overfitting of the model. In v1 the loss values were only slightly better, but visually looking at the gradients of the loss function (printed every 100 iterations in training), it was clear that the final 1000 iterations (from 2000-3000) was the primary area where the collapse was happening.

Table 6: Generator, Discriminator, and Supervised Losses Across Model Versions.

Version	$\mathcal{L}_D$	$\mathcal{L}_G$	$\mathcal{L}_S$
v0	0.0000	0.0004	0.0003
v1	0.0001	0.0006	0.0006
v14	0.0015	0.0019	0.0003

Part of the improvement in loss values observed between versions v1 and v14 was due to the raising of γ from 1 to 5 in v2. γ is just the weighting of supervisor loss relative to the adversarial loss in the generator. By increasing γ, the generator learned to place more emphasis on latent temporal transitions, and less emphasis on adversarial objectives, which helped reduce the collapse of loss values and overfitting of the model.

In v13 I increased the z_dim from 4 to 16. As the model learns from latent reconstructions, adding more dimensions to the noise vectors input to the generator allows for the model to learn more complex, refined relationships (see 4). Though the evaluation metrics nor loss values did not change significantly from the introduction of this change in v13, I decided to leave the higher dimensionality of noise vectors in the v14 version, which may have contributed to its ability to balance latent representations, temporal relationships, and novelty.

C.3 Dataset Size and g2 Test Results

Out of curiosity, I wanted to test the full extent of the limitations of a small dataset. I re-gathered the same macroeconomic datapoints from the World Bank WDI ([11]), but this time for the set of all countries in the world. This global dataset had about 717 unique training windows (many countries were omitted during the data cleaning process due to lack of data points). On the larger dataset the model's performance surprisingly varied widely, and was significantly less accurate in every way than the small training set. As you can see from the graphs of unemployment with time (6) and gdp growth per time step (7), the synthetic data produced by the global model clustered toward extremes. While the tuned model performed clearly better on the small dataset, I would like to do further research into what specifically made the model with the larger training set produce such anomalous results. My inclination is that the adjustments which lead to optimization on a small dataset may

cause either collapse or explosion of loss values on a larger dataset, leading to the model overemphasising the importance of outliers in the original data.

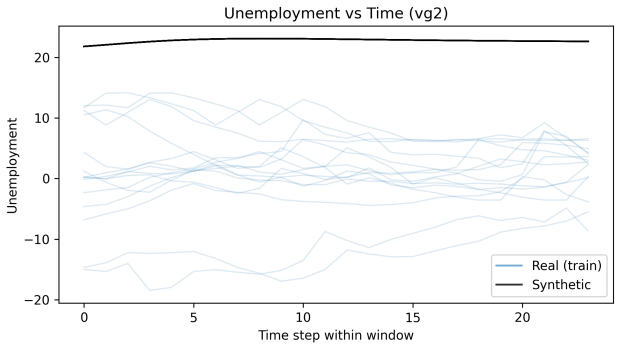

Figure 6: Unemployment Over Time (g2 Synthetic)

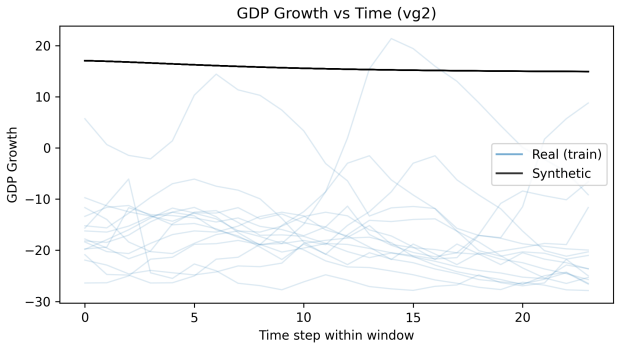

Figure 7: GDP Growth Over Time (g2 Synthetic)

I also think that the amount of sequence overlap in sliding windows–a decision which was necessary for increasing the training steps availability on my small dataset–was ultimately harmful to the global dataset. When there are enough original batches for training without implementing sliding windows, then the inclusion of sliding windows just means that trends will be repeated and perhaps overemphasized in training. Since my original countries were chosen for relative stability in their economies while still representing different geographic areas, it makes sense that there were not many extreme values present in the training data. Using all countries in the world however would expose the model to training data from developing economies which are more susceptible to shocks or high inflation. This could be another reason why extreme points are more prevalent in the synthetic data produced by the global dataset, as the global model would be exposed to many sequences with such inconsistent values.