On the Constrained Time-Series Generation Problem Supplementary Material

A Datasets

In our experiments we consider two publicly available datasets and one synthetic dataset. All the datasets have different characteristics such as periodicity, noise, correlation, and number of features. In particular, the **daily stock** dataset uses daily historical Google stock data from 2004 to 2019 with 6 features, namely *open, high, low, close, adjusted close, and volume*. When running univariate experiments, we only used the **open** feature from the daily-stock dataset. The **energy data** from the UCI Appliances energy prediction dataset Candanedo et al. [2017] contains 28 features, at 10-minute resolution, with noisy periodicity and correlation. Finally, the synthetic **sine** dataset contains multivariate sinusoidal time-series, where each dimension is created independently, sampling the frequencies and phases according the following equation:

$$x_i(t) = sin(2\pi\eta_i t + \theta_i),\ s.t.\ \eta_i \sim \mathcal{U}[0,1] \wedge \theta_i \sim \mathcal{U}[-\pi,\pi],\ \forall i \in \{1,...,5\} \tag{1}$$

This synthetic dataset comes from prior work Yoon et al. [2019]. Where not-otherwise stated, we consider time-series of length 24. In table 1 we summarize the dataset properties, while in Table 15, Table 17 and Table 16 we report all the detailed statistics.

Table 1: Dataset Description

Dataset Name	Data type	Samples	$\dim(\mathbf{x})$	Data Resolution	Link
Stocks	Real	3,773	6	1-day	Link
Energy	Real	19,711	28	10-minutes	Link
Sines	Synthetic	10,000	5	data-point	-

All the datasets are normalized between [-1,1] for the diffusion models.

B Benchmarks

We compare our approaches against existing time-series generative models, i.e., GT-GAN Jeon et al. [2022], TimeGAN Yoon et al. [2019], RCGAN Esteban et al. [2017], C-RNN-GAN Mogren [2016],a Recurrent Neural Networks (RNN) trained with T-Forcing and P-Forcing Lamb et al. [2016], Graves [2013], WaveNET Oord et al. [2016], and WaveGAN Donahue et al. [2018]. We use and modify the publicly available source code for each of the methods:

- GT-GAN Jeon et al. [2022] : openreview
- TimeGAN Yoon et al. [2019] : github
- RCGAN Esteban et al. [2017] : github
- C-RNN-GAN Mogren [2016] : github
- T-Forcing Graves [2013] : github
- P-Forcing Lamb et al. [2016] : github
- WaveNET Oord et al. [2016] : github
- WaveGAN Donahue et al. [2018] :github

In particular for T-forcing and P-forcing we use a 3-layer GRUs with hidden dimensions four times the size of input features, as suggested in Yoon et al. [2019].

For constrained time-series generation scenarios, we restrict our analysis to the top three performing benchmarks, and we adapt their architectures as follows:

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

- **Trend** and **Fixed-Values** constraints. We condition the generators by introducing a new dimension that contains the **trend** or the **fixed-values**. During the training the trend and the fixed-values are extracted directly from the input time-series to let the models rely on this additional information.
- **Hard-Constraints**. We follow the recent work in Di Liello et al. [2020], Xu et al. [2018], Bengio et al. [2017], and we extend the benchmark architectures by introducing a penalty loss. This loss penalizes the generative models proportional to how much the generated time-series violate the input constraint, and it is added to the original model loss by a scale factor λ_c, which we consider as a hyper-parameter tuned in the experiments. Most importantly, for GT-GAN and TimeGAN we add an optimization step, which lasts $1/4$ of the total epochs, and it optimizes the generator alone w.r.t. to the constraint loss. We found that these models benefit from this additional optimization step.

We also employ *rejection-sampling* and *fine-tuning* using COP-method on all their generated synthetic TS.

C Implementation details

We implement our work in Python. Specifically, we use PyTorch Paszke et al. [2019] to implement diffusion models, and we use the Sequential Least Squares Programming (SLSQP) solver Jorge and Stephen [2006] in Scipy's optimization module Virtanen et al. [2020] for COP-method.

All the deep generative models are trained on an NVIDIA T4 GPU, with 4 CPU and 16gb or RAM. To compare the computational times, the inference is done on a 4 CPU 3rd generation AMD EPYC processors for all the models including the COP-method. The default hyper-parameters for the diffusion model are reported in Table 2; we specify in each section when different hyper-parameters are used to compute the results.

Table 2: Diffusion Model default hyper-Parameters

Hyper-parameter	Value	Hyper-parameter	Value
batch-size	16	layers	4
β_1	1.0e-06	learning-rate	0.0001
β_T	0.5	n-heads	8
channels	64	noise-steps T	50
diffusion-embedding-dim	128	noise schedule	quadratic
epochs	10000	weight-decay	1.0e-06
kernel-size	2	constraint	None

D Constrained Optimization Method (COP)

D.1 Algorithm and Details

In Algorithm 1, we present the procedure of the *COP-method* to generate or fine-tune TS such that they conform to constraints. An illustration of the method is shown in Figure 1. We recall that COP frames the task of generating a TS sample as optimizing the value of a set of ordered points that make up the TS sample, such that the sample satisfies domain properties (like auto-correlation). In particular, it starts from an initial sample TS (taken from the dataset, or randomly generated) and optimizes its values according to its objective (e.g., maximize the distance between the generated and initial sample), while respecting some constraints (which could be statistical properties or additional structural constraints). With this problem formulation, we can use existing COP solvers (e.g., SLSQP) to get new samples by solving those non-linear constraints and objectives. Thus the COP solver using the specified constraints and objective function becomes the generative process.

For Algorithm 1, the parameters we used in our experiments are as follows: $b = 0.1$, $\theta_w = 3$, $\theta_r = 0.5$, $\eta_r = 10$, $\eta_i = 2$. If the task is to match a trend, then we set $\theta_w = L$ to be the length of the TS. For very long time series, using a $\theta_w < L$ can help make it easier for the COP solver by breaking

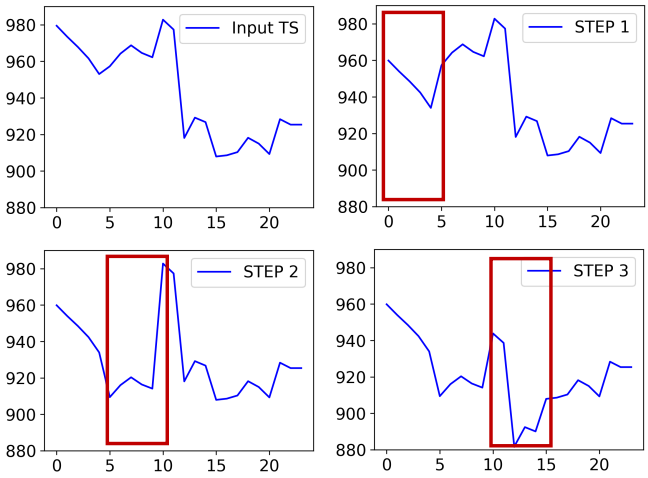


Figure 1: An example of a TS being altered (fine-tuning or generation) by the *COP-method*; changes are made within a window that convolves over the TS. Top-left is the original TS and the window shifts to the right (follow images clockwise) and changes the TS from the starting data in order to respect constraints.

the problem into chunks. However, for some constraints, we may have to solve for all points (i.e. $\theta_w = L$) at once. The initial seed TS ($\mathbf{x}_0$) is a sample from the dataset (Line 6 of Algorithm 1), but we also do experiments with different seed TS, and present the results in Section G.5. With respect to the constraints that we input into the COP-method, they come from the hard constraints in C. The objective used in COP-method for generating TS is to maximize L2-norm of the difference with the seed TS. If a soft-constraint in C is a trend to be followed, then the objective is updated to encourage the COP solver to minimize the L2-norm of the generated TS with the trend; the objective becomes a weighted combination as follows: $f(\hat{\mathbf{x}}, \mathbf{x}, \mathbf{s}) = (1-\omega) * \|\mathbf{x} - \hat{\mathbf{x}}\|_2^2 - \omega * \|\mathbf{s} - \hat{\mathbf{x}}\|_2^2$ where $\omega \in [0, 1]$. By using different ω values one can trade-off between matching the trend and pushing the generated TS to be different from the seed TS. For our experiments that have a trend in constraints C, we set $\omega = 1.0$.
Finally, if COP-method is used to fine-tune a TS to fit constraints, then the objective is changed to minimize L2-norm with the input TS. This is to help prevent changing the TS significantly when fine-tuning. We found L2-norm to work well for our experiments, but other distance functions, like L1 norm or percentage difference, can be used as well.

The constraints used in the COP-method are the same as those input to the other methods, with one exception. We need to additionally constrain the solver to match the statistical properties of real data.

D.2 Realism constraints for Time Series

To give an example of constraints one might like to impose for the realism of a generated dataset, let us consider stock prices TS in **daily stocks** dataset. It is typically required for synthetic stock price TS to preserve stylized facts (a term in economics Sewell [2011]) about the financial markets. These include the distributions of returns, and return auto-correlations Bouchaud et al. [2018], Vyetrenko et al. [2019]. The return at each point in time is defined as the percentage change in the value. For auto-correlation, we use the discrete auto-correlation function for real data. Equations for returns, and autocorrelation (for 1 dimensional TS) are shown in Equation 2 on the left and right side respectively.

$$r_{\mathbf{x}}(t) = \frac{\mathbf{x}_t - \mathbf{x}_{t-1}}{\mathbf{x}_{t-1}}, \quad \rho_{\mathbf{xx}}(\tau) = \frac{E[(\mathbf{x}_{t+\tau} - \mu)(\mathbf{x}_t - \mu)]}{\sigma_{\mathbf{x}}^2} \tag{2}$$

where τ is the auto-correlation lag parameter and μ is the mean value of the TS.

Algorithm 1 *Constrained Optimization Search for TS Generation/Fine-Tuning*

Input: Seed TS $\mathbf{x}_0$, Constraints C , Objective Function f, budgets for constraints b, window size θ_w, window overlap ratio θ_v, number of retries η_r, number of iterations per sample η_i
Output: $\hat{\mathbf{x}}$
len_TS ← get_length($\mathbf{x}_0$)
bounds ← (0.98*min($\mathbf{x}_0$),1.02*max($\mathbf{x}_0$))
for $r \leftarrow 1$ to η_r **do**
  $\hat{\mathbf{x}}_{seed} \leftarrow \mathbf{x}_0$
  { the seed here is a dataset sample, but it can be initialized in different ways; for example, it can also be sampled using brownian motion and rescaled using a sampled real TS' mean and variance}
  $\hat{\mathbf{x}} \leftarrow$ copy($\hat{\mathbf{x}}_{seed}$)
  $W_{pos} \leftarrow$ get_all_window_positions(len_TS,θ_w, θ_v)
  { Window positions with overlap would be (for example) [0, 4], [2, 6], ...}
  $b_r \leftarrow b \times 2^r$
  $C_b \leftarrow$ update_constraints_with_budget(C,b_r)
  for $i \leftarrow 1$ to η_i **do**
    $v_f^* \leftarrow \infty$
    $w_{pos}^* \leftarrow \emptyset$
    $\hat{\mathbf{x}}_{i,best} \leftarrow$ copy($\hat{\mathbf{x}}$)
    for (ϕ_{start},ϕ_{end}) $\in W_{pos}$ **do**
      (status, $\hat{v}_f$, $\hat{\mathbf{x}}_{i,\phi_{start},\phi_{end}}$) ← SLSQP_optimize(f, C_b, $\hat{\mathbf{x}}_i$[ϕ_{start}:ϕ_{end}], bounds)
      {solver returns the updated points. Then we insert them into the candidate solution to get the updated solution}
      $\hat{\mathbf{x}}_{i,temp} \leftarrow \hat{\mathbf{x}}_i$[0:$\phi_{start}$] | $\hat{\mathbf{x}}_{i,\phi_{start},\phi_{end}}$ | $\hat{\mathbf{x}}_i$[ϕ_{end}:]
      if (status = success) & ($\hat{v}_f < v_f^*$) **then**
        $v_f^* \leftarrow \hat{v}_f$
        $\hat{\mathbf{x}}_{i,best} \leftarrow \hat{\mathbf{x}}_{i,temp}$
        $w_{pos}^* \leftarrow$ (ϕ_{start},ϕ_{end})
      end if
    end for
    if $v_f^* \neq \infty$ **then**
      $W_{pos} \leftarrow W_{pos} \backslash w_{pos}^*$
      $\hat{\mathbf{x}} \leftarrow \hat{\mathbf{x}}_{i,best}$
    end if
  end for
  if $\hat{\mathbf{x}} \neq \hat{\mathbf{x}}_{seed}$ **then**
    return $\hat{\mathbf{x}}$
  end if
end for
return $\emptyset$

In the COP method we use auto-correlation of the returns as a constraint – we task the solver to match this property– for the *daily stocks* dataset. For the *Energy* dataset and *Sines* data, we used the autocorrelation of the TS (not the autocorrelation of the returns) as the constraint to improve the realism of the data generated. In our experiments, the auto-correlation lag parameter τ is set to 5. We constrain the solver by taking the L2-norm of the difference in the autocorrelation vectors and limiting that error. The initial budget for this constraint is set to 0.1, which can increase if the solver cannot find a solution (see line 9 in Algorithm 1), and then the solver will iteratively try again.

D.3 WGAN-based Constrained Optimization (WGAN-COP)

One of the main limitations of the presented COP-method is the need to explicitly define all data properties that generated synthetic data must have. In the *COP* method we had to add constraints to match the autocorrelation of the TS signal as a way of capturing desired TS properties. We would ideally like a function $f^* : \mathcal{X} \rightarrow \mathbb{R}$ able to evaluate how much the synthetic data resembles the real

one. However, a single domain-agnostic metric to evaluate synthetic data does not exist yet Alaa et al. [2022].

Following the recent advances in generative adversarial networks however, we notice that the *critic* f_w of a WGAN Arjovsky et al. [2017], Gulrajani et al. [2017] matches the description of our function f^*, in the sense that it will return a higher value if an input sample resembles real data, i.e. it looks like it was sampled from the real distribution of data. The critic also has additional interesting properties, as it is trained using the Wasserstein, or Earth-Mover (EM), distance $W(q, p)$ between the real distribution $q(\mathbf{x})$ and the synthetic one $p(\mathbf{x})$. This distance is continuous everywhere and differentiable almost everywhere under mild assumptions Arjovsky et al. [2017], and the critic f_w must lie within the space of 1-Lipschitz function $||f_w||_L \leq 1$. This means that: 1) we can train the critic till convergence to get a reliable approximation of the Wasserstein distance Arjovsky et al. [2017]; 2) the critic value correlates with sample realism or quality Gulrajani et al. [2017].

Therefore we can first train the WGAN architecture, and then we can replace the constraints used to represent desired TS data properties (e.g. auto-correlation) with the critic function f_w which we put into the objective function. We call this adaptation of COP-method with WGAN as *WGAN-COP*. WGAN-COP tries to maximize f_w while guaranteeing any additional explicit constraints. A COP-solver can get gradients w.r.t. sample $\mathbf{x}$ via back-propagation over the critic's neural network $\nabla_\theta f_w(\mathbf{x})$. In general, this approach does not hold for all the GAN architectures (e.g., those that minimize KL divergence), as the gradients can saturate with no guide for the COP.

Initial experiments show that *WGAN-COP* does not need to explicitly define realism constraints with comparable performance, however the training of the WGAN is in general expensive and unstable, especially with high-dimensional data.

E Diffusion Models

We now introduce the pseudo-code algorithms for the proposed diffusion-based approaches, namely *DiffTime*, *Loss-DiffTime* and *Guided-DiffTime*. For all the models we keep the same choice of diffusion steps $T = 50$, and we compute the noise variance $\boldsymbol{\beta}$ using a quadratic scheduler with a start value of β =1.0e-06 and end value of $\beta_T = 0.5$. We evaluate the impact of different choices of T and $\boldsymbol{\beta}$ in Section G.1; and different noise scheduler in Section G.2. We recall also that $\alpha_t := 1 - \beta_t$ and $\hat{\alpha}_t := \prod_{i=1}^{t} \alpha_i$.

E.1 DiffTime

DiffTime is our base diffusion model approach, which can be trained to incorporate both *trend* and *fixed-values* constraints. The basic model (i.e., without any constraint) follows the standard diffusion model procedures. Algorithm 2 shows *DiffTime* training process, while Algorithm 3 shows the inference process to generate new synthetic time-series.

Algorithm 2 Unconstrained *DiffTime* Training

Input: input TS distribution $q(\mathbf{x}_0)$, number of epochs $\mathbf{E}$
Output: trained diffusion function $\boldsymbol{\epsilon}_\theta$
for $i = 1$ **to** $\mathbf{E}$ **do**
  $t \sim \text{Uniform}(\{1, \dots, T\})$
  $\mathbf{x}_0 \sim q(\mathbf{x}_0)$; $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  Take gradient step on
  $\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\hat{\alpha}_t}\mathbf{x}_0 + \sqrt{1 - \hat{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$
end for

Algorithm 3 Unconstrained *DiffTime* Sampling

Input: trained diffusion function $\boldsymbol{\epsilon}_\theta$
Output: synthetic time-series $\hat{\mathbf{x}}$
$\hat{\mathbf{x}}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $t = \mathbf{T}$ **to** 1 **do**
  $\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\boldsymbol{z} = \mathbf{0}$
  $\hat{\mathbf{x}}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\hat{\mathbf{x}}_t - \frac{1-\alpha_t}{\sqrt{1-\hat{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\hat{\mathbf{x}}_t, t) \right) + \sigma_t \boldsymbol{z}$
end for
Return $\hat{\mathbf{x}}_0$

Trend Constraint. To constrain a particular trend, we condition the diffusion process using a trend time-series $\mathbf{s} \in \chi$. We follow the recent work of Tashiro et al. [2021] to define our conditional diffusion model, and we show the training procedure in Algorithm 4. At each training iteration, a trend $\boldsymbol{s}$, extracted directly from the input time-series $\mathbf{x}_0 \sim q(\mathbf{x}_0)$, and is used to condition the generative model. The trend $\boldsymbol{s}$ can be any interpolation of the input time-series $\mathbf{x}_0$. In our experiments, during training we compute the trend by dividing the time-series in two, and fitting each half with a linear interpolation. We combine the linear interpolations to obtain a very simple trend $\boldsymbol{s}$, and retain the model from just copying the trend. During inference, we test the ability of the model to generalize using instead a low-order (i.e., 3) polynomial approximation. In Figure 2 we show an example of time-series and its trends, used respectively for training and inference, while in Section F.3 we show some examples of generated time-series.

Algorithm 4 Trend-Constrained *DiffTime* Training

Input: input TS distribution $q(\mathbf{x}_0)$, epochs $\mathbf{E}$
Output: trained diffusion function $\boldsymbol{\epsilon}_\theta$
for $i = 1$ **to** $\mathbf{E}$ **do**
  $t \sim \text{Uniform}(\{1, \dots, T\})$
  $\mathbf{x}_0 \sim q(\mathbf{x}_0)$; $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  $\boldsymbol{s}$ = poly-interpolation($\mathbf{x}_0$)
  Take gradient step on
  $\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\hat{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\hat{\alpha}_t}\boldsymbol{\epsilon}, t|\boldsymbol{s}) \right\|^2$
end for

Algorithm 5 Trend-Constrained *DiffTime* Sampling

Input: trained diffusion function $\boldsymbol{\epsilon}_\theta$, trend $\boldsymbol{s}$
Output: synthetic time-series $\hat{\mathbf{x}}$
$\hat{\mathbf{x}}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $t = \mathbf{T}$ **to** 1 **do**
  $\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\boldsymbol{z} = \mathbf{0}$
  $\hat{\mathbf{x}}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\hat{\mathbf{x}}_t - \frac{1-\alpha_t}{\sqrt{1-\hat{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\hat{\mathbf{x}}_t, t|\boldsymbol{s}) \right) + \sigma_t \boldsymbol{z}$
end for
Return $\hat{\mathbf{x}}_0$

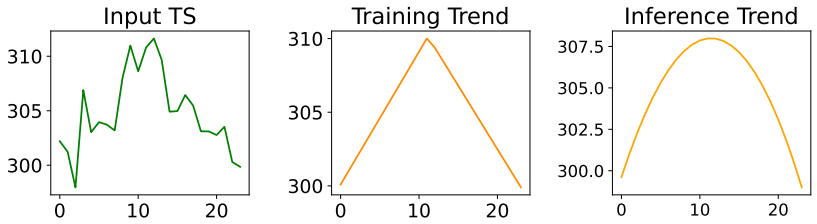


Figure 2: An example of trend-constraints.

Fixed-Value Constraint. To guarantee the *fixed point* constraints, which are hard constraints, we modify the *reverse process* of *DiffTime*, and we explicitly include them in the latent variables $\mathbf{x}_{1:T}$. The *reverse process* is shown in Algorithm 6, while the training procedure remains the standard one, shown in Algorithm 2. The sampling algorithm shows that at each diffusion step t we explicitly enforce the fixed-points values in the noisy time-series $\hat{\mathbf{x}}_t$, such that, $\forall\ r_{i,j} \in \mathcal{R},\ \hat{x}_{i,j} = r_{i,j}$ where $\hat{x}_{i,j} \in \hat{\mathbf{x}}_t$. This approach guarantees that the generated time-series have the desired fixed-point values — in fact the fixed-values are enforced also for $t = 0$ into $\hat{\mathbf{x}}_0$ which is our final synthetic time-series. By enforcing these fixed point at each iteration, we empirically found that the diffusion process better adapts the synthetic time-series to incorporate them. Figure 9 shows some examples of generated time-series with fixed-values constraint.

Algorithm 6 *DiffTime* Sampling - Fixed-Values Constraint

Input: trained diffusion function $\boldsymbol{\epsilon}_\theta$, fixed-points constraints $\mathcal{R}$
Output: synthetic time-series $\hat{\mathbf{x}}$
$\hat{\mathbf{x}}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $t = \mathbf{T}$ **to** 1 **do**
  $\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\boldsymbol{z} = \mathbf{0}$
  $\hat{\mathbf{x}}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\hat{\mathbf{x}}_t - \frac{1-\alpha_t}{\sqrt{1-\hat{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\hat{\mathbf{x}}_t, t) \right) + \sigma_t \boldsymbol{z}$
  for $r_{i,j} \in \mathcal{R}$ **do**
    $\hat{x}_{t-1,i,j} = r_{i,j}$
  end for
end for
Return $\hat{\mathbf{x}}_0$

E.2 Loss-DiffTime

We now discuss how to introduce a loss penalty into the diffusion model presented in the previous section, to incorporate more complex constraints. While the sampling algorithm is the same of Algorithm 3, the training now incorporates a penalty term into the loss function:

$$L(\theta) := \mathbb{E}_{t,\mathbf{x}_0,\boldsymbol{\epsilon}}\left[\left\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right\|^2 + \rho f_c(\hat{\mathbf{x}}_0)\right] \tag{3}$$

where $\mathbf{x}_t = \sqrt{\hat{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\hat{\alpha}_t}\boldsymbol{\epsilon}$ and $\hat{\mathbf{x}}_0 = \frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\hat{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right)$. The function f_c represents any differentiable constraint we want to incorporate.
The training pseudo-code is reported in Algorithm 7.

Algorithm 7 *Loss-DiffTime* Training

Input: input TS distribution $q(\mathbf{x}_0)$, number of epochs $\mathbf{E}$, differentiable constraint f_c, scale parameter ρ
Output: trained diffusion function $\boldsymbol{\epsilon}_\theta$
for $i = 1$ **to E do**
  $t \sim \text{Uniform}(\{1, \dots, T\})$
  $\mathbf{x}_0 \sim q(\mathbf{x}_0)$; $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
  $\mathbf{x}_t = \sqrt{\hat{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\hat{\alpha}_t}\boldsymbol{\epsilon}$
  $\hat{\boldsymbol{\epsilon}} = \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)$
  Take gradient step on
  $\nabla_\theta \left\|\boldsymbol{\epsilon} - \hat{\boldsymbol{\epsilon}}\right\|^2 + \rho f_c\left(\frac{1}{\sqrt{\alpha_t}}(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\hat{\alpha}_t}}\hat{\boldsymbol{\epsilon}})\right)$
end for

E.3 Guided-DiffTime

While *Loss-DiffTime* model is able to incorporate any constraint, it requires to train a new diffusion function $\boldsymbol{\epsilon}_\theta$ for any new constraint. To overcome this limitation, we introduce *Guided-DiffTime* that does not require re-training for new constraints — we train a single unconstrained diffusion model using Algorithm 2 and then we *guide* this model during inference using a differentiable constraint f_c. We show this *guided* sampling procedure in Algorithm 8. In particular, at each diffusion step, we get gradients from the differentiable constraints to guide (condition) the synthetic time-series. The parameter ρ weights the constraint during the generative process.

In Section H.1 we show how *Guided-DiffTime* can dramatically reduce the carbon footprint, by reducing the computational resources needed to handle new constraints.

Algorithm 8 *Guided-DiffTime*

Input: trained diffusion function $\boldsymbol{\epsilon}_\theta$, differentiable constraint $f_c : \mathcal{X} \to \mathbb{R}$, scale parameter ρ
Output: synthetic time-series $\mathbf{x}_0$
$\hat{\mathbf{x}}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
for $t = \mathbf{T}$ **to** 1 **do**
  $\hat{\epsilon} \leftarrow \epsilon_\theta(\hat{\mathbf{x}}_t, t)$
  $\hat{\epsilon} \leftarrow \hat{\epsilon} - \rho\sqrt{1-\hat{\alpha}_t}\,\nabla_{\hat{\mathbf{x}}_t} f_c(\frac{1}{\sqrt{\hat{\alpha}_t}}(\hat{\mathbf{x}}_t - \hat{\epsilon}\sqrt{1-\hat{\alpha}_t}))$
  $\hat{\mathbf{x}}_{t-1} \leftarrow \sqrt{\hat{\alpha}_{t-1}}\left(\frac{\hat{\mathbf{x}}_t - \sqrt{1-\hat{\alpha}_t}\hat{\epsilon}}{\sqrt{\hat{\alpha}_t}}\right) + \sqrt{1-\hat{\alpha}_{t-1}}\hat{\epsilon}$
end for
return $\hat{\mathbf{x}}_0$

E.4 Modelling the diffusion function

We approximate the diffusion function ϵ_θ using a deep neural network, whose architecture is based on the groundbreaking work of Tashiro et al. [2021], Kong et al. [2020]. The architecture is composed by a 1-layer TransformerEncoder Paszke et al. [2019], full-connected and 1d-Convolutional layers. The diffusion steps t are encoded using a 128-dimensions embedding as proposed in previous work Tashiro et al. [2021], Vaswani et al. [2017], Kong et al. [2020]. Figure 3 shows the neural network architecture.

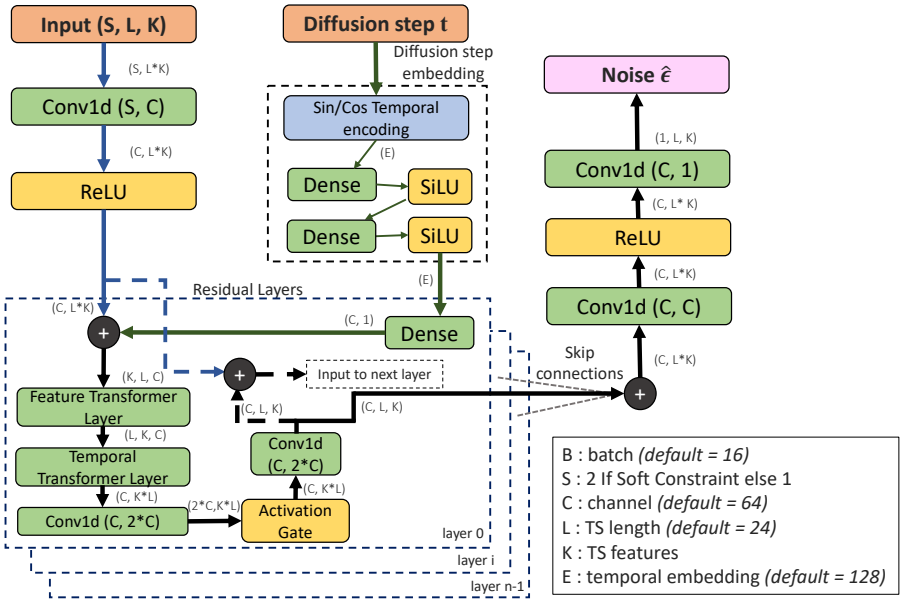


Figure 3: Diffusion model $e_\theta(\mathbf{x}_t, t)$ architecture.

Since our architecture is mainly based on CSDI, we only discuss the main difference with respect to the original work Tashiro et al. [2021]. In particular, we remove the *side information* provided as embedding, and we incorporate all our conditionals along the input time-series $\mathbf{x}$. In fact, our the conditional trend $\mathbf{s}$ has the same shape of the input time-series $\mathbf{x}$. Thus, we can create an input Tensor with K features, L length, and C channels, where the first channel contains the conditional trend $\mathbf{s}$ and the second channel contains the input time-series $\mathbf{x}$. We also change the kernel-size of the convolutional layers, which we found to be an important hyper-parameters to tune according the volatility and length of the input time-series. For example, sine data of length 24 requires a kernel size of 6. Stock data requires kernel size 2 for time-series of length 24, while the kernel size should be increased to 24 for stock time-series with length 360.

For the noise level we use a *Quadratic-Scheduler* which defines β_t as follows:

$$\beta_t = \left(\sqrt{\beta_1} + t \cdot \frac{\sqrt{\beta_T} - \sqrt{\beta_1}}{T-1} \right)^2$$

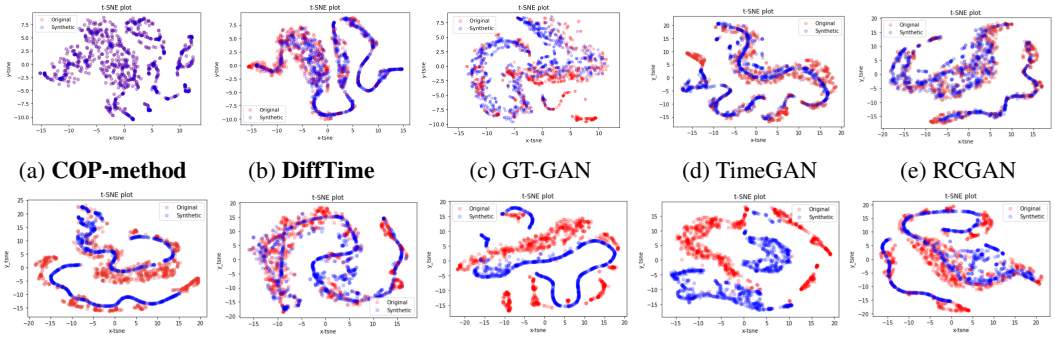
where $T = 50$ are the diffusion steps, and $\beta_1 = 1.0e-06$ and $\beta_T = 0.5$.

F Experimental details and results

In this section, we report additional details about the experiments we show in the main body of the paper.

F.1 Unconstrained generation

In Figure 4 we report the t-SNE analysis for all the approaches, which we omitted due to the limited space, in the main body of the paper. Notice that, to save computational resources, we do not recompute all the approaches but we use results from previous published work Yoon et al. [2019], Jeon et al. [2022] for the same dataset (*daily stocks*). The figure shows again that *DiffTime* and *COP-method* can generate realistic time-series beating existing benchmark algorithms. In particular, the figure shows that our approaches have significantly better performance with better overlap between red and blue samples.



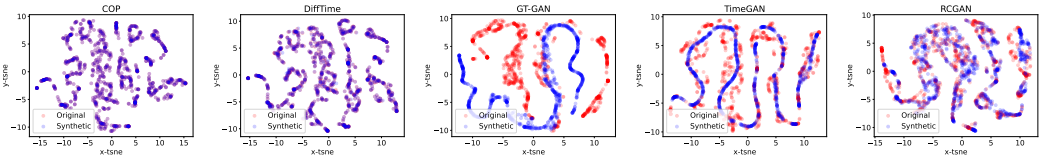
(a) **COP-method** (b) **DiffTime** (c) GT-GAN (d) TimeGAN (e) RCGAN

(f) C-RNN-GAN (g) T-Forcing (h) P-Forcing (i) WaveNET (j) WaveGAN

Figure 4: t-SNE visualizations on multivariate stock data, where a greater overlap of blue and red dots shows a better distributional-similarity between the generated data and original data. Our approaches show the best performance.

F.2 Trend constraint

We report in Figure 5 the t-SNE analysis which we omitted due to space limitations in the main body of the paper. This figure confirms the quantitative evaluation, with *DiffTime* and *COP-method* being the best models also in terms of covering the input distribution — they show better overlap between red and blue dots.



(a) COP-method (b) DiffTime (c) GT-GAN (d) TimeGAN (e) RCGAN

Figure 5: A t-SNE visualizations of *Trend* constrained data, where a greater overlap of blue and red dots implies a better distributional-similarity between the generated data and original data. Our approaches show the best performance.

We report in Figure 6 some example of the generated time-series, showing how our synthetic time-series are closer to the input trend. The figure also shows that *GT-GAN* is only able to generate a very simple time-series matching just the upwards or downwards trend component.

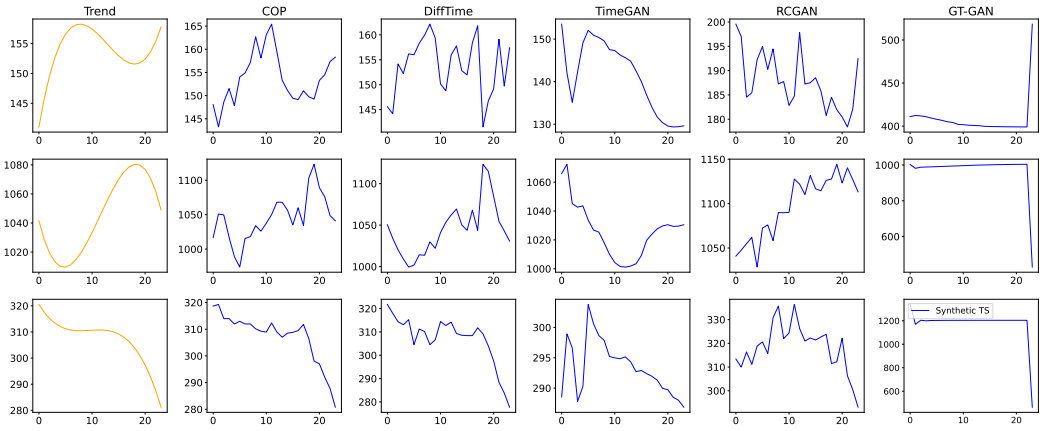


Figure 6: Example of Trend constraints and related synthetic time-series.

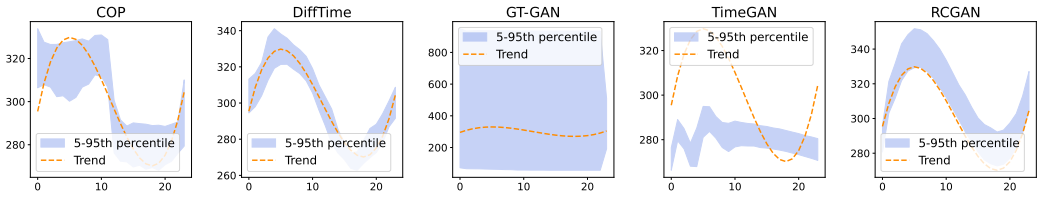
Sinusoidal Trend Finally we evaluate the case of a sinusoidal trend, i.e., the trend is provided as a sine wave, computed similarly to Eq. 1. Considering the peculiar properties of a sinusoidal trend, i.e., its periodicity, we investigate additional metrics, including: **L2 distance** and **DTW distance** which measures how much the synthetic data follows a trend constraint by evaluating the distance between the TS and the trend using L2 norm and Dynamic Time Warping (DTW) approach Berndt and Clifford [1994], respectively; **Fourier distance** which applies a Fourier transformation and compares the basis of the periodic trend and the synthetic TS generated.

Table 3 shows the evaluated quantitative metrics. The table confirms the results shown in the main body of the paper: our approaches achieve the best performance in terms of Discriminative and Predictive score; they also have the closest distance w.r.t. to the input trend. It is interesting to note that the DTW and spectral transformation techniques effectively capture any latent similarity patterns with the trends. For example, the spectral transformation highlights how some methods, like RCGAN, are able to somehow capture the trend even if shifted (which is also visible on Figure 7).

Table 3: Soft Constraints (Sinusoidal Trend) - Time-Series Generation

Algo	Inference-Time	L2 Distance	DTW Distance	Fourier-based distance
COP (Ours)	0.73 ± 0.05	46.3±32.9	35.8±25.8	0.57±0.57
DiffTime (Ours)	0.02 ± 0.00	**35.57±16.99**	**27.57±13.12**	**0.49±0.57**
GT-GAN	**0.00±0.00**	1699.4±1253.1	1692.5±1253.9	1.74±2.51
TimeGAN	**0.00±0.00**	121.35±61.30	87.29±50.25	1.06±1.11
RCGAN	**0.00±0.00**	124.82±83.29	95.73±72.62	0.70±0.75

In Figure 7 we fixed a trend for all the approaches, and we sample 1000 time-series to evaluate the generated time-series. The blue shaded area shows the 5% and 95% percentiles of the generated synthetic time-series.

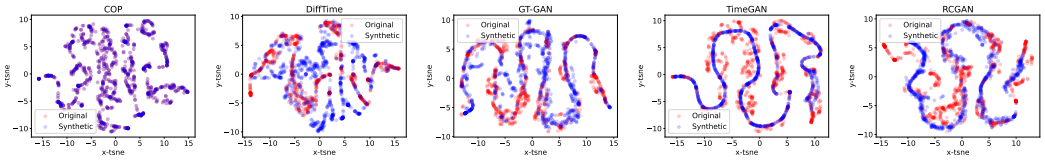


(a) COP-method (b) DiffTime (c) GT-GAN (d) TimeGAN (e) RCGAN

Figure 7: A visualizations of *Trend* constrained data, where the orange dotted time-series is the trend, and the shaded blue area shows the 5% and 95% percentiles of the generated synthetic time-series. Our approaches show the best performance with time-series closer to the input trend.

F.3 Fixed-values constraint

For the fixed-values constraint we consider two fixed-value points at index 6 and 18 of the input time-series, which represent the points at 25% and 75% positions, respectively. We report in Figure 8 the t-SNE analysis which we omitted due to space limitations in the main body of the paper. This figure confirms the quantitative evaluation, with *DiffTime* and *COP-method* being the best models also in terms of covering the input distribution — they show better overlap between red and blue dots. In particular, we recall that while *DiffTime* is not perfectly covering the input distribution, it always guarantee (i.e., 100% of the time) that the synthetic time-series pass through the two input fixed-points.



(a) COP-method (b) DiffTime (c) GT-GAN (d) TimeGAN (e) RCGAN

Figure 8: A t-SNE visualizations of *Fixed-values* constrained data, where a greater overlap of blue and red dots implies a better distributional-similarity between the generated data and original data. Our approaches are among the best models.

We report in Figure 9 some example of the generated time-series. This picture highlights the ability of *DiffTime* to generate reasonable time-series passing through the two fixed-points. COP-method shows the best results in this case, although it doesn't change the TS much from the input TS given to the COP-method. On the other hand, our DiffTime method does a better job of generating more different TS while satisfying the constraints.

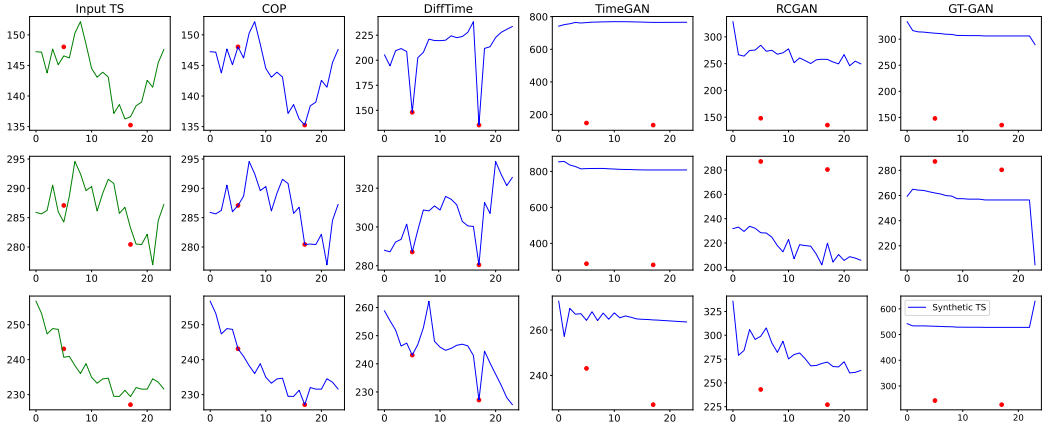
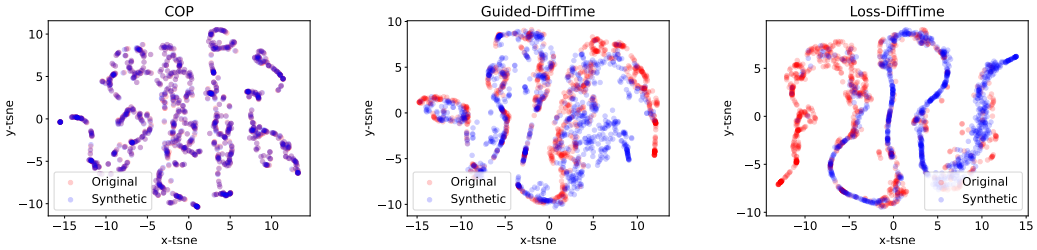


Figure 9: Example of Fixed-values constraint and related synthetic time-series.

F.4 Global Minimum constraint

For the global minimum generation, we enfored the time-series to have a global minimum at index 10. For *Guided-DiffTime* we use $\rho = 2$ while for *Loss-DiffTime* we use $\rho = 3.5$. In Figure 10 we report the t-SNE analysis.



(a) COP-method (b) Guided-DiffTime (c) Loss-DiffTime

(d) GT-GAN (e) TimeGAN (f) RCGAN

Figure 10: A t-SNE visualizations of *Global-Min* constrained data, where a greater overlap of blue and red dots implies a better distributional-similarity between the generated data and original data. Our approaches shows the best performance.

We report in Figure 11 some example of the generated time-series. While most of the approaches generate synthetic time-series that respect the global minimum constraint, our methods better cover the input distribution (see Figure 10), i.e., more fidelity in the generated data. In fact, in all the benchmarks the generated time-series are very similar, while our approaches have more diverse time-series.

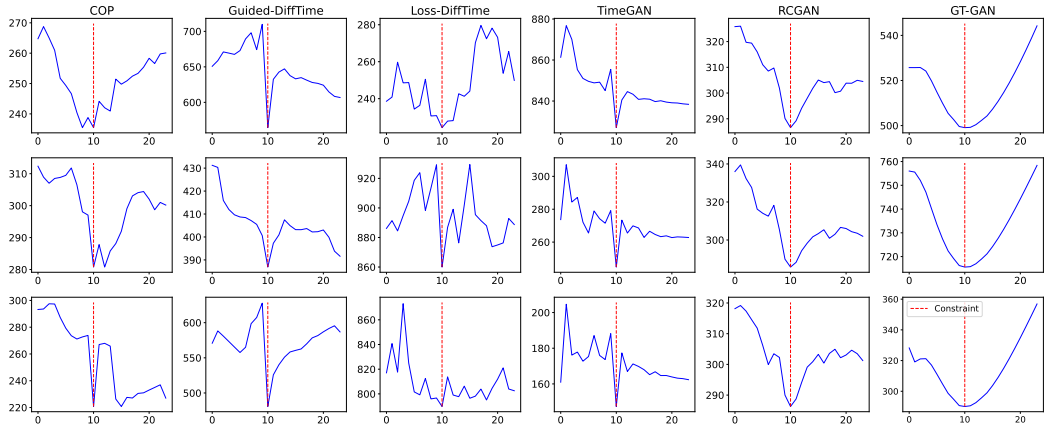
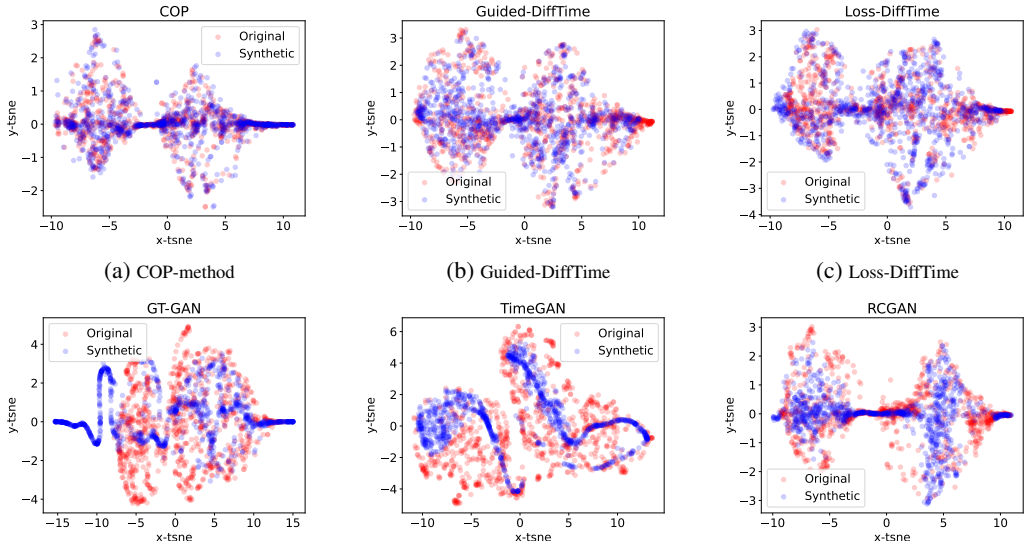


Figure 11: Example of Global-min constraint and related synthetic time-series.

F.5 Multivariate constraint

Finally, we report the multivariate constraint using the multivariate Google stock data. This constraint guarantees a well known financial data property where: the feature *High* has always the highest

value w.r.t. to the other features; and the feature *Low* has always the lowest value w.r.t. to the other features. For *Guided-DiffTime* we use $\rho = 0.001$ while for *Loss-DiffTime* we use $\rho = 3.5$. We report in Figure 12 the t-SNE analysis which we omitted due to space limitations in the main body of the paper. This figure confirms the quantitative evaluation, with *DiffTime* and *COP-method*, and our approaches show a better coverage of the input distribution, with a higher overlap between red and blue dots.



(a) COP-method (b) Guided-DiffTime (c) Loss-DiffTime

(d) GT-GAN (e) TimeGAN (f) RCGAN

Figure 12: A t-SNE visualizations of *Multivariate* constrained data, where a greater overlap of blue and red dots implies a better distributional-similarity between the generated data and original data. Our approaches shows the best performance.

We report in Figure 13 some example of the generated time-series. In this case, it's worth noticing that *Guided-DiffTime* and *COP-method* have among the best performance, showing time-series that respect the multivariate constraints (i.e., high feature has always the maximum value, while low feature is the lowest). The Figure also shows that the generated time-series from the GT-GAN have not exactly the common statistical properties of stock data Bouchaud et al. [2018]; while RCGAN and TimeGAN have a huge difference between High and Low features, which is unlikely in real data and in the training set.

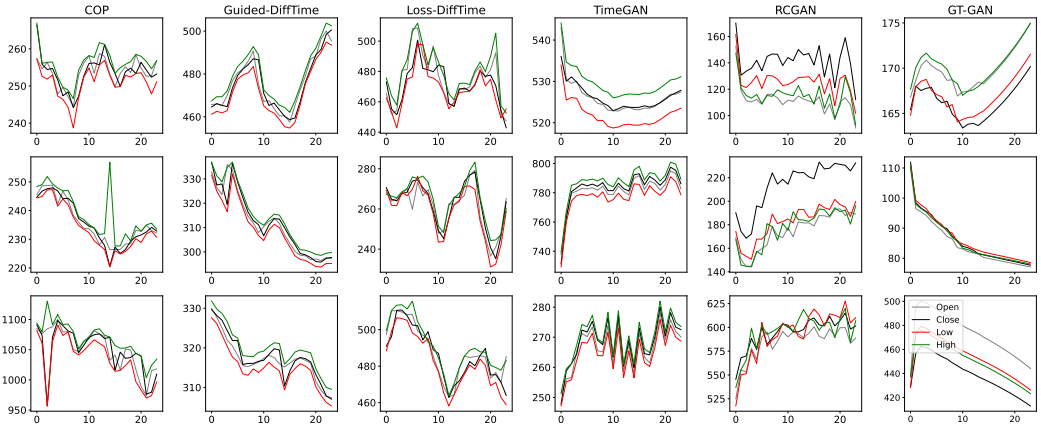


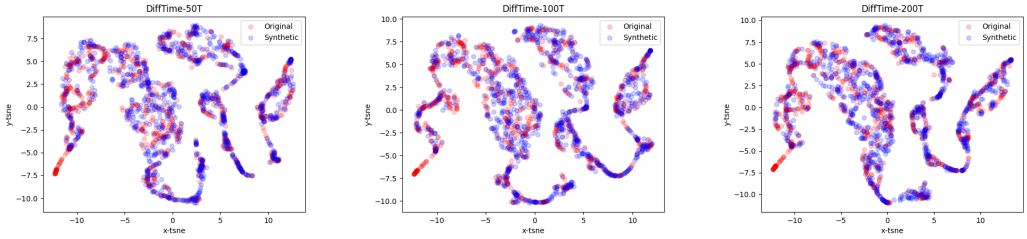
Figure 13: Example of multivariate constraint and related synthetic time-series.

G Ablation Study

In this section we carried out an ablation study of the proposed approaches. Where not otherwise stated, we consider univariate stock-data.

G.1 Diffusion steps

Here we evaluate the impact of a different number of diffusion steps in the diffusion models. We vary the diffusion steps using $T \in [50, 100, 200]$. Figure 14 shows the t-SNE comparison for the different diffusion steps, which show all the same performance. Therefore, in all our experiments we considered the most economic setup of $T = 50$. In table 4 we evaluate the impact of the different diffusion steps in the model according the quantitative metrics. Also in this table, we notice that the increasing the diffusion steps do not improve the results.



(a) DiffTime $T = 50$ (b) DiffTime $T = 100$ (c) DiffTime $T = 200$

Figure 14: A t-SNE visualizations of *DiffTime* for different diffusion steps $T \in [50, 100, 200]$.

Table 4: *DiffTime* with different diffusion steps $T \in [50, 100, 200]$.

Algo	Discr-Score	Pred-Score	Inference-Time
DiffTime $T = 50$	**0.05±0.03**	**0.21±0.00**	**0.020±0.00**
DiffTime $T = 100$	0.07±0.02	0.22±0.00	0.049±0.02
DiffTime $T = 200$	0.06±0.01	**0.21±0.00**	0.091±0.01

G.2 Noise Variance

Here we evaluate the impact of a different noise variance scheduler in the diffusion models. We recall that we consider $T = 50$ diffusion steps, and we set the minimum noise level $\beta_1 = 1.0e - 06$, the maximum level to $\beta_T = 0.5$. Following recent work in diffusion models Tashiro et al. [2021], Nichol and Dhariwal [2021], Song et al. [2021], we define β_t by consider the following schedulers:

• *Linear-Scheduler*:

$$\beta_t = \left(\beta_1 + t \cdot \frac{\beta_T - \beta_1}{T - 1}\right)$$

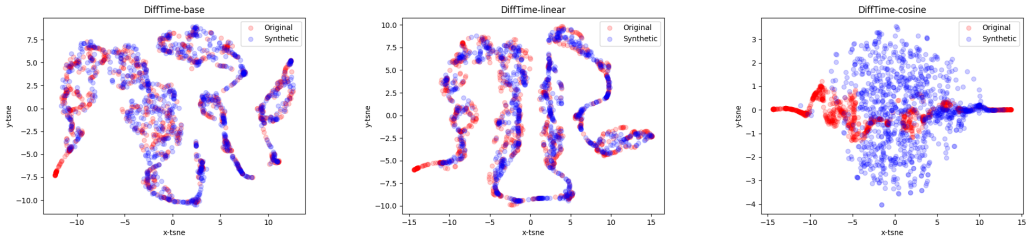
• *Quadratic-Scheduler*:

$$\beta_t = \left(\sqrt{\beta_1} + t \cdot \frac{\sqrt{\beta_T} - \sqrt{\beta_1}}{T - 1}\right)^2$$

• *Cosine-Scheduler*:

$$\beta_t = \beta_1 + (\beta_T - \beta_1) \cdot \frac{1}{2}\left(1 + cos\left(\frac{\pi * t}{T}\right)\right)$$

In Figure 15 we show the t-SNE comparison for the different schedulers. The figure shows that in our case *Cosine* scheduler does not achieve a good performance, while both linear and quad scheduler better cover the input data distribution.



(a) DiffTime-Quad (b) DiffTime-Linear (c) DiffTime-Cosine

Figure 15: A t-SNE visualizations of *DiffTime* for different noise variance scheduler.

Table 5: *DiffTime* with different noise variance scheduler.

Algo	Discr-Score	Pred-Score	Inference-Time
DiffTime-quad	**0.05±0.03**	**0.21±0.00**	**0.021±0.00**
DiffTime-linear	0.06±0.02	**0.21±0.00**	**0.021±0.01**
DiffTime-cosine	0.25±0.02	0.23±0.00	**0.021±0.00**

G.3 Diffusion model architecture

We now evaluate the impact of different model layers, and hyper-parameters, on DiffTime performance. We introduce the following variants of DiffTime:

- *DiffTime-K-Heads* - we change the number of *attention heads*, from 1 to 8;
- *DiffTime-LSTM* - we replace the convolutional layers using recurrent layers (i.e., LSTM) along the attention mechanism, which is particularly successful for imputation and interpolation of TS Shukla and Marlin [2020];
- *DiffTime-full-LSTM* - we replace all the convolutional and transformer layers by using LSTM layers, which is common for time-series generation Mogren [2016];
- *DiffTime-full-CNN* - we replace the transformer layers using convolutional layers;

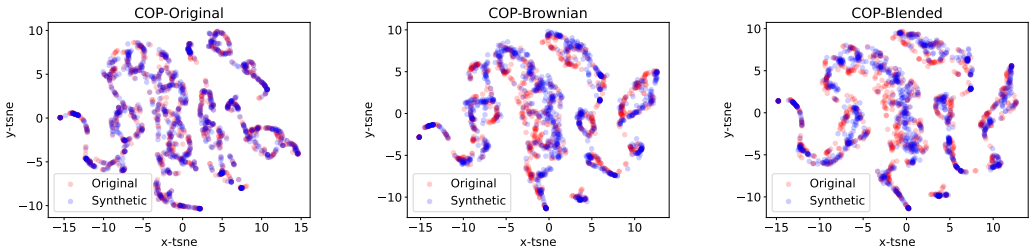
Table 6: *DiffTime* using different layers and hyper-parameters

Algo	Discr-Score	Pred-Score	Inference-Time
DiffTime-1Heads	**0.03±0.02**	**0.21±0.00**	**0.02±0.01**
DiffTime-4Heads	0.06 ± 0.02	**0.21±0.00**	0.04±0.01
DiffTime-8Heads	0.05 ± 0.03	**0.21±0.00**	**0.02±0.01**
DiffTime-LSTM	0.06 ± 0.01	**0.21±0.00**	**0.02±0.01**
DiffTime-full-LSTM	0.50 ± 0.00	**0.21±0.00**	**0.02±0.01**
DiffTime-full-CNN	0.14 ± 0.04	**0.21±0.00**	0.03±0.01

The results are shown in Table 6. The table highlights the performance of the current architecture, which uses transformer and convolutional layers. Moreover, the table shows that the number of attention heads should be tuned according to the input dataset to achieve better results.

G.4 COP-method Initial seed

Here we evaluate the impact of different initial seed into COP-method framework. We test the following: a) the input time-series distribution $q(\mathbf{x})$; b) Brownian random noise that is scaled to a real TS sample; c) *Blended* time-series where we add brownian noise to the input time-series from $q(\mathbf{x})$. Figure 16 shows the t-SNE results which show that all the different approaches achieve realistic results, covering the input data distribution. Quantitative metrics are shown in Table 7, and confirm the applicability of COP-method to the different input seed data.



(a) COP-Original (b) COP-Brownian (c) COP-Blended

Figure 16: A t-SNE visualizations *COP-method* using different input seed data.

Table 7: *COP-method* with different input seed data.

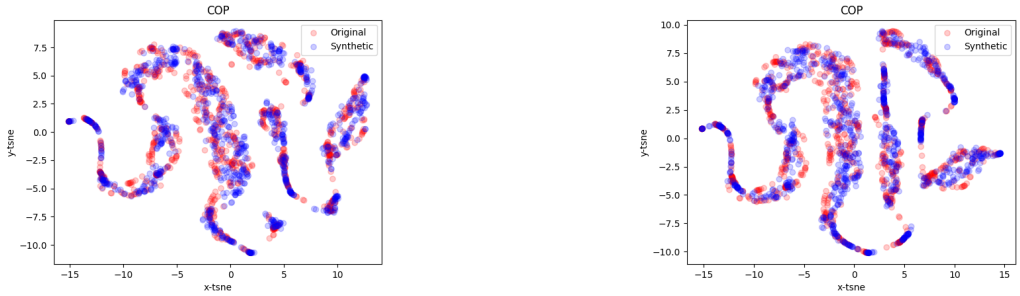
Algo	Discr-Score	Pred-Score	Inference-Time
COP-Blended	**0.01±0.01**	**0.20±0.00**	0.81±0.02
COP-Brownian	0.02±0.02	**0.20±0.00**	0.70±0.05
COP-Original	0.02±0.01	**0.20±0.00**	0.63±0.01

G.5 COP performance using different distance metrics

COP maximizes a L2 distance as objective, to obtain diversity and create new synthetic samples starting from the input initial seeds. However, L2 distance may not necessarily be the best proxy for diversity, and we can use other distance-based metrics. In this ablation experiment, we compare the performance of COP comparing two different distance metrics. In particular, we empirically evaluated L2 distance and L1 distance. Figure 17 shows that both the distance metrics preserve distributional similarity in the synthetic data, which we empirically evaluated using t-SNE. However, the L2 distance achieves slightly better quantitative results, a shown in Table 8.

Table 8: COP using L1 vs L2 distance to generate synthetic samples.

Algo	Discr-Score	Pred-Score
COP L2-distance	**0.017±0.006**	**0.203±0.001**
COP L1-distance	0.021±0.012	**0.203±0.002**

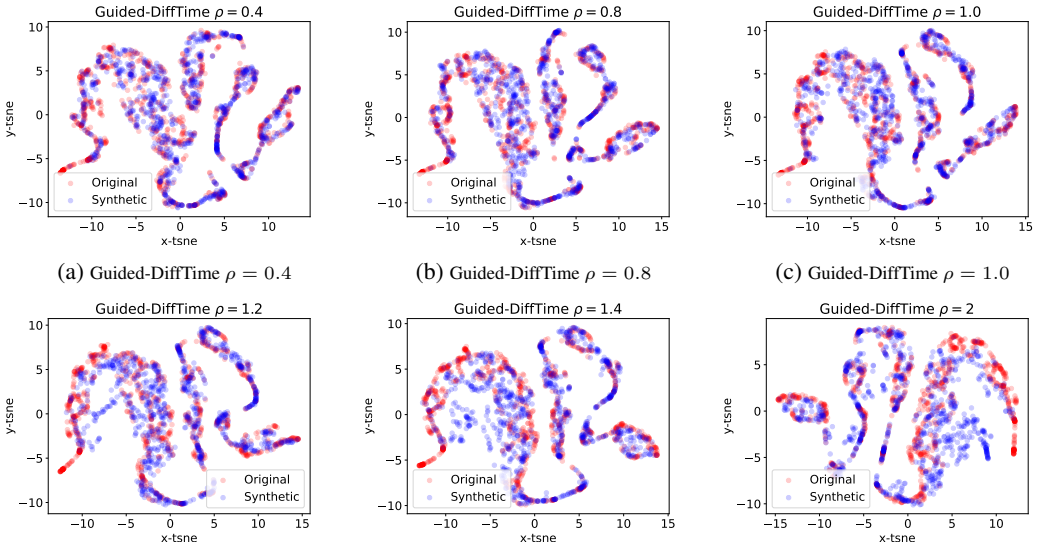


(a) L2-distance (b) L1-distance

Figure 17: A t-SNE visualizations of the time-series generated by COP by maximizing the L2 or L1 distance w.r.t. initial seed samples.

G.6 The impact of the scale parameter ρ to *Guided-DiffTime*

We evaluate the impact of the scale parameter ρ to the *Guided-DiffTime* when applied to *Global Min* constraint. In Figure 18 we report the t-SNE analysis, while in Figure 19 we show some examples of generated synthetic time-series. The quantitative metrics are reported in Table 9. It's worth noticing that (as expected) increasing of the scale parameter ρ, results in the model trading-off realism to guarantee the constraints for all the synthetic time-series.



(a) Guided-DiffTime $\rho = 0.4$ (b) Guided-DiffTime $\rho = 0.8$ (c) Guided-DiffTime $\rho = 1.0$

(d) Guided-DiffTime $\rho = 1.2$ (e) Guided-DiffTime $\rho = 1.4$ (f) Guided-DiffTime $\rho = 2.0$

Figure 18: A t-SNE visualizations of *Global Min* constrained data at varying of the scale parameter ρ for *Guided-DiffTime*.

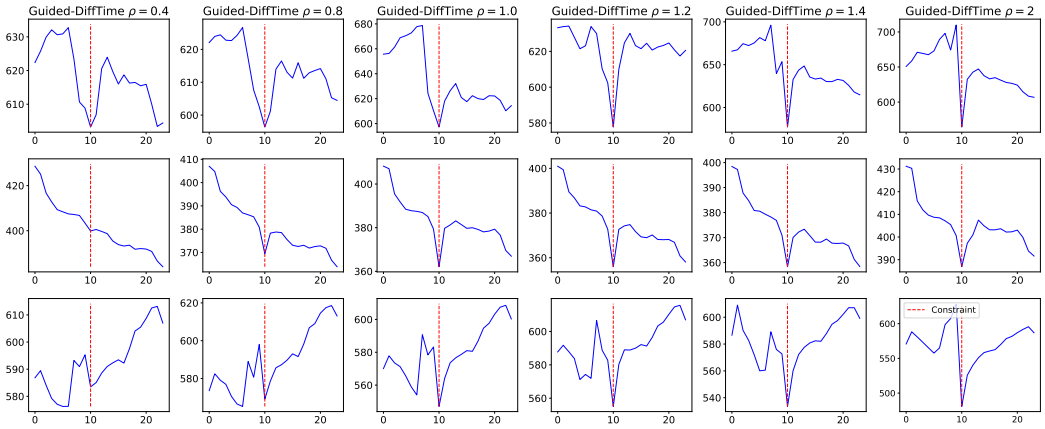


Figure 19: Example of *Global Min* constrained time-series at varying of the scale parameter ρ for *Guided-DiffTime*.

Table 9: *Guided-DiffTime* global-min constrained generation at varying of ρ.

Algo GuidedDiffTime	Discr-Score	Pred-Score	Inference-Time	Satisfaction Rate
ρ =0.4	**0.04±0.03**	**0.21±0.00**	0.034±0.02	0.36±0.00
ρ =0.8	**0.04±0.03**	**0.21±0.00**	0.033±0.02	0.70±0.00
ρ =1.0	0.05±0.02	**0.21±0.00**	**0.032±0.01**	0.81±0.00
ρ =1.2	0.06±0.02	**0.21±0.00**	**0.032±0.01**	0.88±0.00
ρ =1.4	0.06±0.03	**0.21±0.00**	0.033±0.04	0.90±0.00
ρ =2.0	0.07±0.02	**0.21±0.00**	0.034±0.02	**0.94±0.00**

H Additional Experiments

In this section we present additional experiments which we omitted in the main body of the paper due to limited space. Where not otherwise stated, we consider univariate stock-data.

H.1 The computational cost of constrained-generation

First we evaluate the impact of adding a new constraint on the proposed models. We evaluate this in terms of computational cost, i.e., the computational resources and time needed to incorporate the new constraints and sample $N = 1000$ time-series for each constraint. For this experiment, we compute the *Global Min* constraint and vary the global minimum index $i \in [0, 23]$, i.e., along all the time-series. Therefore, we have 24 different constraints.

In Table 10[1] we show the training, inference and total computation times required for all the 24 constraints. The table shows that *COP-method* does not require any training, however has a large sampling (inference) time, due to the complexity of the optimization problem. Instead, *Guided-DiffTime* only requires that we train a single unconstrained *DiffTime* model used to handle all the different constraints. Therefore, *Guided-DiffTime* has a very low computational cost with respect to other approaches that have to be re-trained for each new constraint. The table shows that *Guided-DiffTime* is estimated to reduce the emission of around 60% w.r.t. to *COP-method* and around 92% w.r.t. other Deep Generative models. All the deep generative models are trained on a NVIDIA T4 GPU, with 4 cores and 16gb or RAM. To compare the computational times, the inference is done on a 4 core 3rd generation AMD EPYC processors for all the models including COP. Experiments were conducted using AWS cloud service in Ohio region, where the total emissions are estimated using a Machine Learning Impact calculator presented in Lacoste et al. [2019].

Table 10: Constrained Generation - Estimated total computational cost

Algorithm	Training-Time (hrs)	Inference-Time (hrs)	Total-Time (hrs)	Emissions (kgCO_2eq)
COP-method	**0.0**	127.8	127.8	1.25
Guided-DiffTime	12.0	0.2	**12.2**	**0.52**
Loss-DiffTime	312.0	0.1	312.1	12.45
TimeGAN	400.0	**0.0**	400.0	15.96
RCGAN	156.0	**0.0**	156.0	6.22
GT-GAN	192.0	**0.0**	192.0	7.66

[1]The presented values are estimated using available experimental data, to reduce the computational cost.

H.2 Longer time-series using DiffTime

We now evaluate the impact of the different time-series lengths on the generative models for un-constrained generation. Notice that, while this is not the goal of our work, DiffTime and COP shows consistently higher performance for longer time-series, while maintaining a stable training/inference procedure. On the other hand, GANs-based methods, which have inherently unstable training, show decreased performance for longer time-series. We consider daily stock-data with three different lengths $\in [36, 72, 360]$ (i.e., days). For these experiments we keep all the same hyper-parameters and we only change:

- the kernel-size ks of CNN layers in the diffusion model, being $ks \in [3, 6, 24]$ for the different lengths $\in [36, 72, 360]$, respectively;
- the hidden-dimension of *TimeGAN*, *RCGAN*, and *GT-GAN*, which is set to be the time-series length, as suggested by authors and empirically evaluated;
- the window size θ_w of COP, being the time-series length divided by 2.

As mentioned, we found that the training time highly increase for *TimeGAN* and *GT-GAN*, especially with time series of length equal to 360. For *RCGAN* and *DiffTime* the training time is only slightly increased.

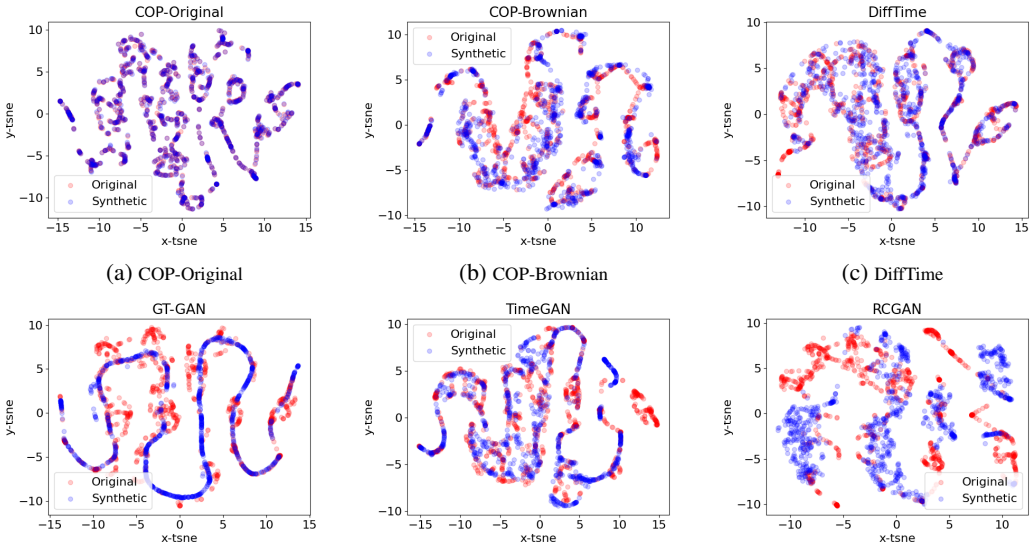
Table 11: Un-Constrained Generation - Longer TS

Length TS	Algo	Discr-Score	Pred-Score	Inference-Time
36	COP-Brownian	**0.01±0.01**	**0.20±0.00**	0.33±0.00
	COP-Original	**0.01±0.01**	**0.20±0.00**	0.04±0.00
	DiffTime	0.04±0.03	0.21±0.00	0.05±0.00
	GT-GAN	0.03±0.02	0.21±0.00	0.00±0.00
	RCGAN	**0.01±0.01**	**0.20±0.00**	0.00±0.00
	TimeGAN	0.03±0.02	**0.20±0.00**	0.00±0.00
72	COP-Brownian	**0.02±0.01**	**0.21±0.00**	0.63±0.00
	COP-Original	**0.02±0.01**	**0.21±0.00**	0.07±0.00
	DiffTime	0.04±0.02	0.22±0.00	0.15±0.00
	GT-GAN	0.09±0.05	0.22±0.00	0.00±0.00
	RCGAN	0.03±0.02	**0.21±0.00**	0.00±0.00
	TimeGAN	0.06±0.02	0.24±0.00	0.00±0.00
360	COP-Brownian	0.06±0.04	**0.20±0.00**	2.39±0.00
	COP-Original	**0.03±0.01**	**0.20±0.00**	0.38±0.00
	DiffTime	0.06±0.06	**0.20±0.00**	0.04±0.00
	GT-GAN	0.18±0.05	**0.20±0.00**	0.03±0.00
	RCGAN	0.09±0.06	0.21±0.00	0.00±0.00
	TimeGAN	0.10±0.09	0.22±0.00	0.00±0.00

The quantitative metrics are reported in Table 11. It's worth noticing that (as expected) increasing the length of the time-series results in lower performance, as the models have more difficulty to capture the longer statistical properties of the time-series. However, *DiffTime* and *COP* have the lower degradation: the Discr. Score of *DiffTime* and *COP* increases only of 0.02 when the time-series length increases from 36 to 360, while the other methods have at least 0.08 (400% more) increase in Discr. Score.

In Figure 20 we report the t-SNE analysis for length 36, while in Figure 21 we show some examples of generated synthetic time-series, normalized w.r.t. their first values. While the generated time-series in Figure 21 may seem reasonable, some of them exhibit very unusual volatility (e.g., RCGAN and TimeGAN generate time-series with more than 30% price changes in 36 days), while others samples have not much diversity (i.e., first two time-series generated by GT-GAN). Importantly, it is also the case that professional traders can easily distinguish between real stock price series and synthetic price series generated by simple price models Mandelbrot and Hudson [2010].

In the next section we better investigate some specific financial properties, called stylized facts Vyetrenko et al. [2020], to show that our approaches outperform the benchmarks in preserving real data properties.



(a) COP-Original (b) COP-Brownian (c) DiffTime

(d) GT-GAN (e) TimeGAN (f) RCGAN

Figure 20: A t-SNE visualization of un-constrained time-series with length equal to 36. A greater overlap of blue and red dots implies a better distributional-similarity between the generated data and original data. Our approaches shows the best performance.

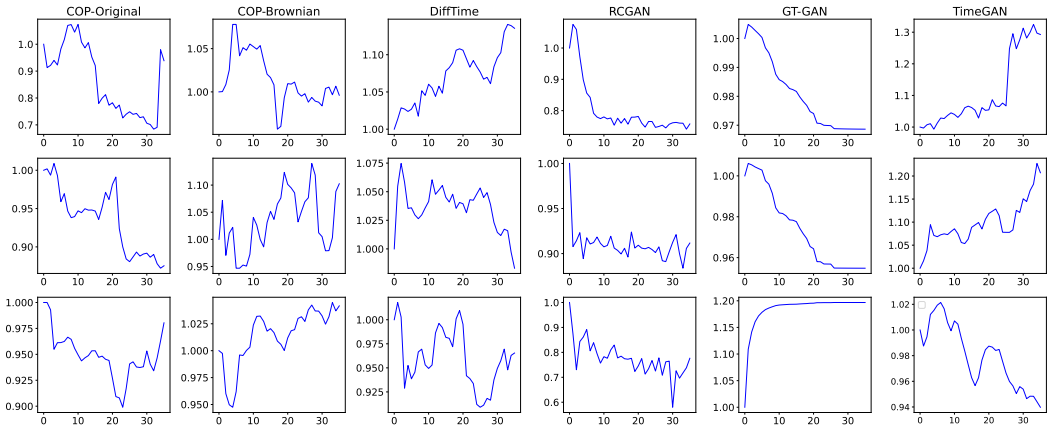


Figure 21: Example of un constrained time-series with length 36.

H.3 Financial properties

In this section we investigate three specific financial properties of price series, showing that synthetic time-series generated by our approaches better preserve such properties w.r.t. existing benchmarks. For example, as asset daily returns usually have fat tail distribution and long-range dependence, we expected the same properties (or *stylized facts*) from artificial markets. To have a fair comparison, we choose the case of time-series with length equal to 36, as the existing benchmarks have the closest performance to us when the length is 36 (seeTable 11).

We evaluate the following three stylized facts *auto-correlations*, *heavy tails distribution*, and *long range dependence*, to evaluate asset return properties. We refer the reader to the work in Vyetrenko et al. [2020] and Bouchaud et al. [2018] for a more detailed introduction to stylized facts.

The first Figure 22 shows the return distribution of real and synthetic time-series, for all the approaches. Our approaches show better overlap between orange and blue distributions, as the synthetic time-series better resemble the real data returns. Is it interesting to note that RCGAN synthetic data has a too much fat-tailed distribution, although in table 11 it has among the best performance in Discr. Score.

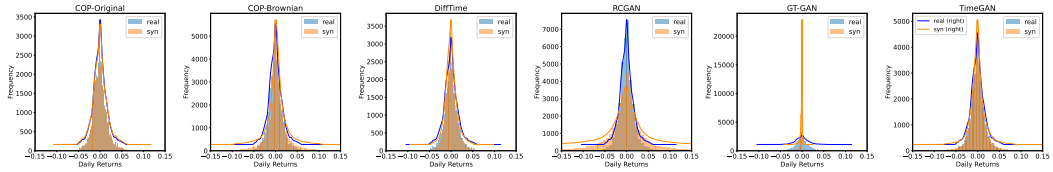


Figure 22: Returns distribution of un-constrained time-series with length 36.

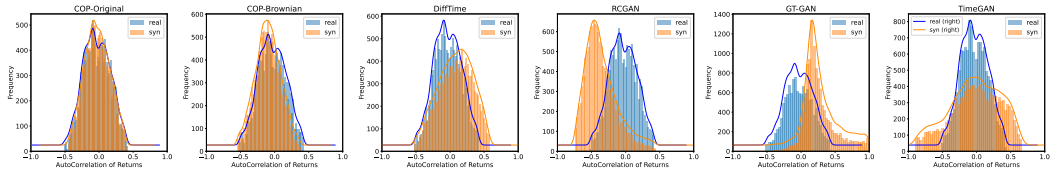


Figure 23: Auto-correlation distribution of un-constrained time-series with length 36.

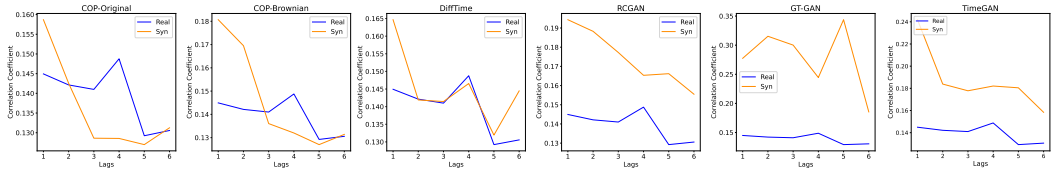


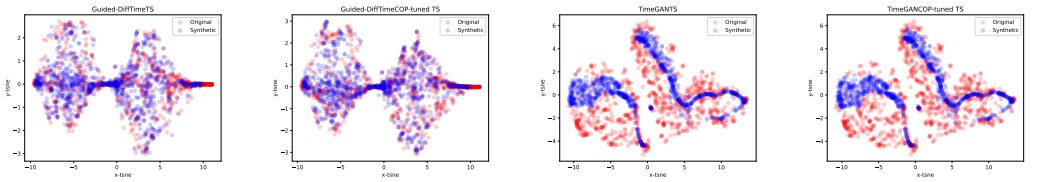
Figure 24: Volatility-clustering and long-range dependence for time-series with length 36.

Figure 23 confirms the superiority of our approaches as the auto-correlation of synthetic returns have much more similarity to those of real data: our approaches show better overlap between orange and blue distributions. Finally, in Figure 24 we show the long-range correlation/dependence of returns, with different lags from 1 to 6 days. The charts show that the volatility decays at increasing number of the days, and that DiffTime has the best performance in preserving this property: orange and blue lines are closer.

H.4 Time-Series fine-tuning using COP

In this section we show that COP can be used to fine-tune synthetic samples and enforce constraints, for any deep learning model. In particular, we recall that COP can take as input synthetic samples that do not respect a given constraint, and it can slightly alter them (see Algorithm 1 and Figure 1) to meet the required properties and comply with the input constraint. We consider again the multivariate constraint of Section F.5, using the multivariate Google stock data. Notice that, COP fine-tuning procedure minimizes the L2 distance between the input samples and the generated ones, i.e., it minimizes the number of changes needed to satisfy the constraints.

In Table 12 we show that COP can fine-tune generated samples and highly improve the percentage of TS that respect the input OHLC constraint. Notice that, COP does not guarantee 100% of satisfaction rate, as for some samples it is not able to guarantee the constraints (under current settings) without destroying original data properties (e.g., autocorrelation), thus it fails. However, COP almost doubles the satisfaction rate, and with different settings it can guarantee even higher satisfaction rate. In particular, while Guided-DiffTime and TimeGAN achieve a satisfaction rate of 72% and 51%, respectively, after the fine-tuning they achieve 97.3% and 89.7%. Importantly, Figure 25 confirms that the data distribution learn by the model is not highly affected by COP fine-tuning procedure.



(a) Guided-DiffTime (b) COP fine-tuned Guided-DiffTime (c) TimeGAN (d) COP fine-tuned TimeGAN

Figure 25: A t-SNE visualizations of time-series generated by Guided-DiffTime and TimeGAN for OHLC, before and after applying COP for fine-tuning. The fine-tuning does not alter or deteriorate the data distribution.

Table 12: OHLC-Constrained Generation

Algo	COP Fine-Tuning	Satisfaction-Rate
Guided-DiffTime	False	0.72±0.02
TimeGAN	False	0.51±0.02
Guided-DiffTime	True	**0.97±0.03**
TimeGAN	True	**0.90±0.02**

I Extended related work comparison

In recent years, there has been a growing body of research dedicated to the exploration of synthetic data, with particular emphasis on its application within the financial and healthcare domain van Breugel et al. [2023], Yoon et al. [2019], Jeon et al. [2022], Esteban et al. [2017], Mogren [2016], Coletta et al. [2022], Chen et al. [2021], Coletta et al. [2021]. This surge in interest can be attributed to the escalating utility demonstrated by synthetic data across a diverse array of studies, particularly in scenarios where access to genuine data is restricted due to privacy constraints Alaa et al. [2022], van Breugel and van der Schaar [2023], Coletta et al. [2023], Esteban et al. [2017].

In this section we survey additional related work for synthetic time-series generation. In particular, we consider the following state-of-art approaches: COSCI-GAN Seyfi et al. [2022], RTSGAN Pei et al. [2021], and LS4 Zhou et al. [2023]. COSCI-GAN is a promising GAN-based approach that focuses mostly on synthetic multivariate time series, which originates from a single source (i.e., biometric measurements from a medical patient; or open-high-low-close time-series from financial markets). We consider such work as it shows promising results, especially for the preservation of inter-channel/feature dynamics: we may expect such work to easily capture the OHLC constraint from data itself. The second work, namely RTSGAN, focuses on real-world time series, where sequences can have variable lengths, missing data, and noisy observations. The work proposes a novel generative framework where an encoder-decoder module learns a mapping between a time series instance and a fixed dimension latent vector, and the generative model works on such lower dimensional latent space. To the best of our knowledge, this work shows state-of-art results on multivariate stock data. Finally, LS4 is a generative model that uses latent variables evolving according to a state space ODE to increase modeling capacity. However, differently from us, it focuses on long-sequence modelling and continuous time-series. Therefore, we do not consider this last work as benchmark in our extended comparison.

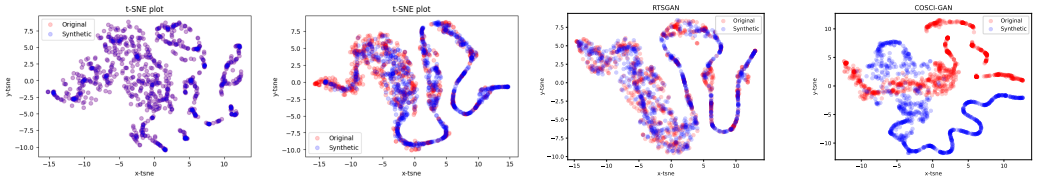
Furthermore, we recall that none of the above mentioned models directly support constrained generation. Thus, we first consider them within the domain of unconstrained time-series (TS) generation. Then, we modified the training procedure of such models by introducing a penalty loss, which penalizes the generative models proportional to how much the generated time-series violate the input constraint. For the constrained generation we specifically focus on the Open-High-Low-Close (OHLC) constraint. We chose OHLC constraint for comparing the new baselines since COSCI-GAN is intended for multivariate time series and OHLC is a constraint on the relative values between 4 time series. For both COSCI-GAN and RTSGAN we follow the official authors' implementation.

I.1 Un-Constrained Generation

We first focus on uncontrained time-series scenarios for multivariate stock-data. We report the quantitative metrics, Discr. and Pred. Score, in Table 13 , while t-SNE analysis is shown in Figure 26. From the results, COP still shows the best distributional similarity w.r.t. to real data, which is empirically evaluated in the t-SNE plot, where blue and red dots almost always overlap. RTSGAN achieves notable performance in terms of Discr. and Pred. scores, with good distributional similarity in the t-SNE chart. However, with respect to properties pertinent to financial data introduced in Section H.3, RTSGAN shows higher autocorrelation than real data, potentially stemming from multiple GRU layers (see Figure 27); and more shallow return distribution.

Table 13: Unconstrained Time-Series Generation - Stock data

Algo	Discr-Score	Pred-Score	Inference-Time
COP (Ours)	.050 ± .017	.041 ± .001	1.01 ± 0.00
DiffTime (Ours)	.097 ± .016	.038 ± .001	0.02 ± 0.00
COSCI-GAN	.412 ± .002	.088 ± .000	**0.00±0.00**
RTSGAN	**.024 ± .007**	**.036 ± .000**	**0.00±0.00**



(a) COP (b) DiffTime (c) RTSGAN (d) COSCI-GAN

Figure 26: A t-SNE visualizations of unconstrained time-series generation. Our models show among the best performance.

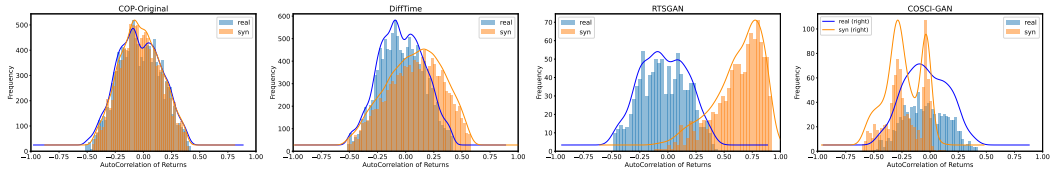


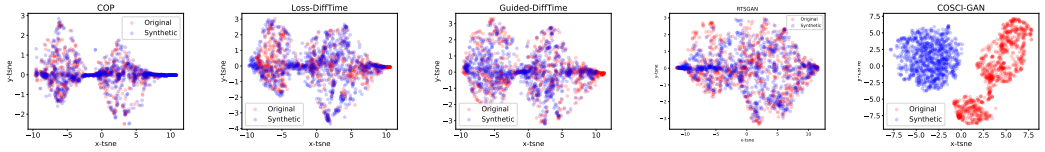
Figure 27: Autocorrelation of returns distribution for un-constrained time-series.

I.2 OHLC-Constrained Generation

We now focus on OHLC constrained time-series scenarios for multivariate stock-data. Table 14 shows the quantitative results. Figure 28 shows the distributional similarity of the new approaches, empirically evaluated through the t-SNE plot. From the results we can observe similar performance as in the unconstrained setting for COSCI-GAN and RTSGAN, both in terms of distributional similarity, discr. and pred. scores. However, looking at the satisfaction rate (i.e., percentage of time-series respecting the input constraint), our methods outperform the two benchmarks. Most importantly, our Guided-DiffTime model stands out for its remarkable capacity to accommodate new constraints without any retraining, constituting a fundamental innovative contribution to the literature on generating TS data.

Table 14: OHLC Constrained - Stock data

Algo	Discr-Score	Pred-Score	Inference-Time	Satisfaction Rate
COP (Ours)	0.04 ± 0.02	**0.04±0.00**	2.17 ± 0.10	**1.00±0.00**
GuidedDiffTime (Ours)	0.08 ± 0.00	**0.04±0.10**	0.15 ± 0.00	0.72 ± 0.02
LossDiffTime (Ours)	0.35 ± 0.04	**0.04±0.01**	0.14 ± 0.00	0.69 ± 0.01
COSCI-GAN	0.45 ± 0.01	0.09 ± 0.00	**0.00±0.00**	0.02 ± 0.00
RTSGAN	**0.02±0.01**	**0.04±0.00**	**0.00±0.00**	0.54 ± 0.02



(a) COP (b) Loss-DiffTime (c) Guided-DiffTime (d) RTSGAN (e) COSCI-GAN

Figure 28: A t-SNE visualizations of OHLC constrained time-series generation. Our models show among the best performance.

Disclaimer

This paper was prepared for informational purposes by the Artificial Intelligence Research group of JPMorgan Chase & Coand its affiliates ("JP Morgan"), and is not a product of the Research Department of JP Morgan. JP Morgan makes no representation and warranty whatsoever and disclaims all liability, for the completeness, accuracy or reliability of the information contained herein. This document is not intended as investment research or investment advice, or a recommendation, offer or solicitation for the purchase or sale of any security, financial instrument, financial product or service, or to be used in any way for evaluating the merits of participating in any transaction, and shall not constitute a solicitation under any jurisdiction or to any person, if such solicitation under such jurisdiction or to such person would be unlawful.

Data Description

We now report all the statistical properties of used datasets.

Table 15: Stock Dataset (GOOG)

Feature	mean	std	min	25%	50%	75%	max
Open	453.23	305.02	49.27	233.25	306.95	621.22	1271.00
High	457.33	307.45	50.54	235.40	309.35	627.55	1273.89
Low	448.81	302.55	47.67	230.75	304.51	612.40	1249.02
Close	453.15	305.13	49.68	233.44	306.44	622.69	1268.33
Adj-Close	453.15	305.13	49.68	233.44	306.44	622.69	1268.33
Volume	7391935.77	8197565.12	7900.00	1959200.00	4674500.00	9723900.00	82768100.00

Table 16: Synthetic Dataset (Sine)

Feature	mean	std	min	25%	50%	75%	max
Sine-1	0.49	0.32	0.0	0.20	0.46	0.80	1.0
Sine-2	0.50	0.32	0.0	0.20	0.46	0.80	1.0
Sine-3	0.50	0.32	0.0	0.19	0.46	0.81	1.0
Sine-4	0.50	0.32	0.0	0.20	0.46	0.80	1.0
Sine-5	0.49	0.32	0.0	0.19	0.46	0.80	1.0

Table 17: Energy Dataset

Feature	mean	std	min	25%	50%	75%	max
Appliances	97.69	102.52	10.00	50.00	60.00	100.00	1080.00
lights	3.80	7.94	0.00	0.00	0.00	0.00	70.00
T1	21.69	1.61	16.79	20.76	21.60	22.60	26.26
RH_1	40.26	3.98	27.02	37.33	39.66	43.07	63.36
T2	20.34	2.19	16.10	18.79	20.00	21.50	29.86
RH_2	40.42	4.07	20.46	37.90	40.50	43.26	56.03
T3	22.27	2.01	17.20	20.79	22.10	23.29	29.24
RH_3	39.24	3.25	28.77	36.90	38.53	41.76	50.16
T4	20.86	2.04	15.10	19.53	20.67	22.10	26.20
RH_4	39.03	4.34	27.66	35.53	38.40	42.16	51.09
T5	19.59	1.84	15.33	18.28	19.39	20.62	25.80
RH_5	50.95	9.02	29.82	45.40	49.09	53.66	96.32
T6	7.91	6.09	-6.06	3.63	7.30	11.26	28.29
RH_6	54.61	31.15	1.00	30.02	55.29	83.23	99.90
T7	20.27	2.11	15.39	18.70	20.03	21.60	26.00
RH_7	35.39	5.11	23.20	31.50	34.86	39.00	51.40
T8	22.03	1.96	16.31	20.79	22.10	23.39	27.23
RH_8	42.94	5.22	29.60	39.07	42.38	46.54	58.78
T9	19.49	2.01	14.89	18.00	19.39	20.60	24.50
RH_9	41.55	4.15	29.17	38.50	40.90	44.34	53.33
T_out	7.41	5.32	-5.00	3.67	6.92	10.41	26.10
Press_mm_hg	755.52	7.40	729.30	750.93	756.10	760.93	772.30
RH_out	79.75	14.90	24.00	70.33	83.67	91.67	100.00
Windspeed	4.04	2.45	0.00	2.00	3.67	5.50	14.00
Visibility	38.33	11.79	1.00	29.00	40.00	40.00	66.00
Tdewpoint	3.76	4.19	-6.60	0.90	3.43	6.57	15.50
rv1	24.99	14.50	0.01	12.50	24.90	37.58	50.00
rv2	24.99	14.50	0.01	12.50	24.90	37.58	50.00

References

Ahmed Alaa, Boris Van Breugel, Evgeny S Saveliev, and Mihaela van der Schaar. How faithful is your synthetic data? sample-level metrics for evaluating and auditing generative models. In *International Conference on Machine Learning*, pages 290–306. PMLR, 2022.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *International conference on machine learning*, pages 214–223. PMLR, 2017.

Yoshua Bengio, Ian Goodfellow, and Aaron Courville. *Deep learning*, volume 1. MIT press Cambridge, MA, USA, 2017.

Donald J Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *Proceedings of the 3rd international conference on knowledge discovery and data mining*, pages 359–370, 1994.

Jean-Philippe Bouchaud, Julius Bonart, Jonathan Donier, and Martin Gould. *Trades, quotes and prices: financial markets under the microscope*. Cambridge University Press, 2018.

Luis M Candanedo, Véronique Feldheim, and Dominique Deramaix. Data driven prediction models of energy use of appliances in a low-energy house. *Energy and buildings*, 140:81–97, 2017.

Richard J Chen, Ming Y Lu, Tiffany Y Chen, Drew FK Williamson, and Faisal Mahmood. Synthetic data in machine learning for medicine and healthcare. *Nature Biomedical Engineering*, 5(6): 493–497, 2021.

Andrea Coletta, Matteo Prata, Michele Conti, Emanuele Mercanti, Novella Bartolini, Aymeric Moulin, Svitlana Vyetrenko, and Tucker Balch. Towards realistic market simulations: a generative adversarial networks approach. In *Proceedings of the Second ACM International Conference on AI in Finance*, pages 1–9, 2021.

Andrea Coletta, Aymeric Moulin, Svitlana Vyetrenko, and Tucker Balch. Learning to simulate realistic limit order book markets from data as a world agent. In *Proceedings of the Third ACM International Conference on AI in Finance*, pages 428–436, 2022.

Andrea Coletta, Svitlana Vyetrenko, and Tucker Balch. K-SHAP: Policy clustering algorithm for anonymous multi-agent state-action pairs. In *Proceedings of the 40th International Conference on Machine Learning*, pages 6343–6363. PMLR, 23–29 Jul 2023. URL `https://proceedings.mlr.press/v202/coletta23a.html`.

Luca Di Liello, Pierfrancesco Ardino, Jacopo Gobbi, Paolo Morettin, Stefano Teso, and Andrea Passerini. Efficient generation of structured objects with constrained adversarial networks. *Advances in neural information processing systems*, 33:14663–14674, 2020.

Chris Donahue, Julian McAuley, and Miller Puckette. Adversarial audio synthesis. *arXiv preprint arXiv:1802.04208*, 2018.

Cristóbal Esteban, Stephanie L Hyland, and Gunnar Rätsch. Real-valued (medical) time series generation with recurrent conditional gans. *arXiv preprint arXiv:1706.02633*, 2017.

Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. *Advances in neural information processing systems*, 30, 2017.

Jinsung Jeon, Jeonghak Kim, Haryong Song, Seunghyeon Cho, and Noseong Park. Gt-gan: General purpose time series synthesis with generative adversarial networks. *Advances in Neural Information Processing Systems*, 35:36999–37010, 2022.

Nocedal Jorge and J Wright Stephen. Numerical optimization, 2006.

Zhifeng Kong, Wei Ping, Jiaji Huang, Kexin Zhao, and Bryan Catanzaro. Diffwave: A versatile diffusion model for audio synthesis. *arXiv preprint arXiv:2009.09761*, 2020.

Alexandre Lacoste, Alexandra Luccioni, Victor Schmidt, and Thomas Dandres. Quantifying the carbon emissions of machine learning. *arXiv preprint arXiv:1910.09700*, 2019.

Alex M Lamb, Anirudh Goyal ALIAS PARTH GOYAL, Ying Zhang, Saizheng Zhang, Aaron C Courville, and Yoshua Bengio. Professor forcing: A new algorithm for training recurrent networks. *Advances in neural information processing systems*, 29, 2016.

Benoit B Mandelbrot and Richard L Hudson. *The (mis) behaviour of markets: a fractal view of risk, ruin and reward*. Profile books, 2010.

Olof Mogren. C-rnn-gan: Continuous recurrent neural networks with adversarial training. *arXiv preprint arXiv:1611.09904*, 2016.

Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *International Conference on Machine Learning*, pages 8162–8171. PMLR, 2021.

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

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, 2019.

Hengzhi Pei, Kan Ren, Yuqing Yang, Chang Liu, Tao Qin, and Dongsheng Li. Towards generating real-world time series data. In *2021 IEEE International Conference on Data Mining (ICDM)*, pages 469–478. IEEE, 2021.

Martin Sewell. Characterization of financial time series. *Rn*, 11(01):01, 2011.

Ali Seyfi, Jean-Francois Rajotte, and Raymond Ng. Generating multivariate time series with common source coordinated gan (cosci-gan). *Advances in Neural Information Processing Systems*, 35: 32777–32788, 2022.

Satya Narayan Shukla and Benjamin Marlin. Multi-time attention networks for irregularly sampled time series. In *International Conference on Learning Representations*, 2020.

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *International Conference on Learning Representations*, 2021.

Yusuke Tashiro, Jiaming Song, Yang Song, and Stefano Ermon. CSDI: Conditional score-based diffusion models for probabilistic time series imputation. *Advances in Neural Information Processing Systems*, 34:24804–24816, 2021.

Boris van Breugel and Mihaela van der Schaar. Beyond privacy: Navigating the opportunities and challenges of synthetic data. *arXiv preprint arXiv:2304.03722*, 2023.

Boris van Breugel, Zhaozhi Qian, and Mihaela van der Schaar. Synthetic data, real errors: how (not) to publish and use synthetic data. *arXiv preprint arXiv:2305.09235*, 2023.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

Svitlana Vyetrenko, David Byrd, Nick Petosa, Mahmoud Mahfouz, Danial Dervovic, Manuela Veloso, and Tucker Hybinette Balch. Get real: Realism metrics for robust limit order book market simulations, 2019.

Svitlana Vyetrenko, David Byrd, Nick Petosa, Mahmoud Mahfouz, Danial Dervovic, Manuela Veloso, and Tucker Hybinette Balch. Get real: Realism metrics for robust limit order book market simulations. In *ACM International Conference on AI in Finance (ICAIF)*, 2020.

Jingyi Xu, Zilu Zhang, Tal Friedman, Yitao Liang, and Guy Broeck. A semantic loss function for deep learning with symbolic knowledge. In *International conference on machine learning*, pages 5502–5511. PMLR, 2018.

Jinsung Yoon, Daniel Jarrett, and Mihaela Van der Schaar. Time-series generative adversarial networks. *Advances in neural information processing systems*, 32, 2019.

Linqi Zhou, Michael Poli, Winnie Xu, Stefano Massaroli, and Stefano Ermon. Deep latent state space models for time-series generation. In *International Conference on Machine Learning*, pages 42625–42643. PMLR, 2023.