A PROOFS

Theorem A.1 (Denoising score matching on time-series)**.** *$l_1(n,s)$ can be replaced by the following $l_2(n,s)$:*

$$l_2(n,s) = \mathbb{E}_{\mathbf{x}_n^0}\mathbb{E}_{\mathbf{x}_{1:n}^s}\left[\left\|M_\theta(s,\mathbf{x}_{1:n}^s,\mathbf{x}_{1:n-1}^0) - \nabla_{\mathbf{x}_{1:n}^s}\log p(\mathbf{x}_{1:n}^s|\mathbf{x}_{1:n}^0)\right\|_2^2\right], \tag{15}$$

where $\mathbf{x}_n^0$ and $\mathbf{x}_{1:n}^s$ are sampled from $p(\mathbf{x}_n^0|\mathbf{x}_{1:n-1}^0)$ and $p(\mathbf{x}_{1:n}^s|\mathbf{x}_{1:n}^0)$. Therefore, we can use an alternative objective, $L_2 = \mathbb{E}_s\mathbb{E}_{\mathbf{x}_{1:N}}\left[\sum_{n=1}^N \lambda(s) l_2(n,s)\right]$ instead of L_1

Proof. At first, if $n = 1$, it can be substituted with the naive denoising score loss by Vincent (2011) since $\mathbf{x}_0^0 = \mathbf{0}$.

Next, let us consider $n > 1$. $l_1(n,s)$ can be decomposed as follows:

$$\begin{aligned} l_1(n,s) = -2\cdot\mathbb{E}_{\mathbf{x}_{1:n}^s}\langle M_\theta(s,\mathbf{x}_{1:n}^s,\mathbf{x}_{1:n-1}^0), \nabla_{\mathbf{x}_{1:n}^s}\log p(\mathbf{x}_{1:n}^s|\mathbf{x}_{1:n-1}^0)\rangle \\ +\mathbb{E}_{\mathbf{x}_{1:n}^s}\left[\left\|M_\theta(s,\mathbf{x}_{1:n}^s,\mathbf{x}_{1:n-1}^0)\right\|_2^2\right] + C_1 \end{aligned} \tag{16}$$

Here, C_1 is a constant that does not depend on the parameter θ, and $\langle\cdot,\cdot\rangle$ means the inner product. Then, the first part's expectation of the right-hand side can be expressed as follows:

$$\begin{aligned} &\mathbb{E}_{\mathbf{x}_{1:n}^s}[\langle M_\theta(s,\mathbf{x}_{1:n}^s,\mathbf{x}_{1:n-1}^0), \nabla_{\mathbf{x}_{1:n}^s}\log p(\mathbf{x}_{1:n}^s|\mathbf{x}_{1:n-1}^0)\rangle] \\ &= \int_{\mathbf{x}_{1:n}^s}\langle M_\theta(s,\mathbf{x}_{1:n}^s,\mathbf{x}_{1:n-1}^0), \nabla_{\mathbf{x}_{1:n}^s}\log p(\mathbf{x}_{1:n}^s|\mathbf{x}_{1:n-1}^0)\rangle \mathrm{p}(\mathbf{x}_{1:n}^s|\mathbf{x}_{1:n-1}^0) d\mathbf{x}_{1:n}^s \\ &= \int_{\mathbf{x}_{1:n}^s}\langle M_\theta(s,\mathbf{x}_{1:n}^s,\mathbf{x}_{1:n-1}^0), \frac{1}{\mathrm{p}(\mathbf{x}_{1:n-1}^0)}\frac{\partial \mathrm{p}(\mathbf{x}_{1:n}^s,\mathbf{x}_{1:n-1}^0)}{\partial \mathbf{x}_{1:n}^s}\rangle d\mathbf{x}_{1:n}^s \\ &= \int_{\mathbf{x}_n^0}\int_{\mathbf{x}_{1:n}^s}\langle M_\theta(s,\mathbf{x}_{1:n}^s,\mathbf{x}_{1:n-1}^0), \frac{1}{\mathrm{p}(\mathbf{x}_{1:n-1}^0)}\frac{\partial \mathrm{p}(\mathbf{x}_{1:n}^s,\mathbf{x}_{1:n-1}^0,\mathbf{x}_n^0)}{\partial \mathbf{x}_{1:n}^s}\rangle d\mathbf{x}_{1:n}^s d\mathbf{x}_n^0 \\ &= \int_{\mathbf{x}_n^0}\int_{\mathbf{x}_{1:n}^s}\langle M_\theta(s,\mathbf{x}_{1:n}^s,\mathbf{x}_{1:n-1}^0), \frac{\partial \mathrm{p}(\mathbf{x}_{1:n}^s|\mathbf{x}_{1:n}^0))}{\partial \mathbf{x}_{1:n}^s}\rangle \frac{\mathrm{p}(\mathbf{x}_{1:n-1}^0,\mathbf{x}_n^0)}{\mathrm{p}(\mathbf{x}_{1:n-1}^0)} d\mathbf{x}_{1:n}^s d\mathbf{x}_n^0 \\ &= \int_{\mathbf{x}_n^0}\int_{\mathbf{x}_{1:n}^s}\langle M_\theta(s,\mathbf{x}_{1:n}^s,\mathbf{x}_{1:n-1}^0), \frac{\partial \mathrm{p}(\mathbf{x}_{1:n}^s|\mathbf{x}_{1:n}^0)}{\partial \mathbf{x}_{1:n}^s}\rangle \mathrm{p}(\mathbf{x}_n^0|\mathbf{x}_{1:n-1}^0) d\mathbf{x}_{1:n}^s d\mathbf{x}_n^0 \\ &= \mathbb{E}_{\mathbf{x}_n^0}\left[\int_{\mathbf{x}_{1:n}^s}\langle M_\theta(s,\mathbf{x}_{1:n}^s,\mathbf{x}_{1:n-1}^0), \frac{\partial \mathrm{p}(\mathbf{x}_{1:n}^s|\mathbf{x}_{1:n}^0)}{\partial \mathbf{x}_{1:n}^s}\rangle d\mathbf{x}_{1:n}^s\right] \\ &= \mathbb{E}_{\mathbf{x}_n^0}\left[\int_{\mathbf{x}_{1:n}^s}\langle M_\theta(s,\mathbf{x}_{1:n}^s,\mathbf{x}_{1:n-1}^0), \nabla_{\mathbf{x}_{1:n}^s}\log p(\mathbf{x}_{1:n}^s|\mathbf{x}_{1:n}^0)\rangle \mathrm{p}(\mathbf{x}_{1:n}^s|\mathbf{x}_{1:n}^0) d\mathbf{x}_{1:n}^s\right] \\ &= \mathbb{E}_{\mathbf{x}_n^0}\mathbb{E}_{\mathbf{x}_{1:n}^s}[\langle M_\theta(s,\mathbf{x}_{1:n}^s,\mathbf{x}_{1:n-1}^0), \nabla_{\mathbf{x}_{1:n}^s}\log p(\mathbf{x}_{1:n}^s|\mathbf{x}_{1:n}^0)\rangle] \end{aligned} \tag{17}$$

Similarly, the second part's expectation of the right-hand side can be rewritten as follows:

$$
\begin{aligned}
&\mathbb{E}_{\mathbf{x}^s_{1:n}}[\left\|M_\theta(s,\mathbf{x}^s_{1:n},\mathbf{x}^0_{1:n-1})\right\|_2^2] \\
&= \int_{\mathbf{x}^s_{1:n}} \left\|M_\theta(s,\mathbf{x}^s_{1:n},\mathbf{x}^0_{1:n-1})\right\|_2^2 \cdot \mathrm{p}(\mathbf{x}^s_{1:n}|\mathbf{x}^0_{1:n-1}) d\mathbf{x}^s_{1:n} \\
&= \int_{\mathbf{x}^0_n}\int_{\mathbf{x}^s_{1:n}} \left\|M_\theta(s,\mathbf{x}^s_{1:n},\mathbf{x}^0_{1:n-1})\right\|_2^2 \cdot \frac{\mathrm{p}(\mathbf{x}^s_{1:n},\mathbf{x}^0_{1:n-1},\mathbf{x}^0_n)}{\mathrm{p}(\mathbf{x}^0_{1:n-1})} d\mathbf{x}^s_{1:n} d\mathbf{x}^0_n \\
&= \int_{\mathbf{x}^0_n}\int_{\mathbf{x}^s_{1:n}} \left\|M_\theta(s,\mathbf{x}^s_{1:n},\mathbf{x}^0_{1:n-1})\right\|_2^2 \cdot \mathrm{p}(\mathbf{x}^s_{1:n}|\mathbf{x}^0_{1:n}) \frac{\mathrm{p}(\mathbf{x}^0_{1:n-1},\mathbf{x}^0_n)}{\mathrm{p}(\mathbf{x}^0_{1:n-1})} d\mathbf{x}^s_{1:n} d\mathbf{x}^0_n \\
&= \int_{\mathbf{x}^0_n}\int_{\mathbf{x}^s_{1:n}} \left\|M_\theta(s,\mathbf{x}^s_{1:n},\mathbf{x}^0_{1:n-1})\right\|_2^2 \cdot \mathrm{p}(\mathbf{x}^s_{1:n}|\mathbf{x}^0_{1:n}) \mathrm{p}(\mathbf{x}^0_n|\mathbf{x}^0_{1:n-1}) d\mathbf{x}^s_{1:n} d\mathbf{x}^0_n \\
&= \mathbb{E}_{\mathbf{x}^0_n}\mathbb{E}_{\mathbf{x}^s_{1:n}}[\left\|M_\theta(s,\mathbf{x}^s_{1:n},\mathbf{x}^0_{1:n-1})\right\|_2^2]
\end{aligned}
\tag{18}
$$

Finally, by using above results, we can derive following result:

$$
\begin{aligned}
l_1 &= \mathbb{E}_{\mathbf{x}^0_n}\mathbb{E}_{\mathbf{x}^s_{1:n}}\left[\left\|M_\theta(s,\mathbf{x}^s_{1:n},\mathbf{x}^0_{1:n-1})\right\|_2^2\right] + C_1 \\
&- 2\cdot \mathbb{E}_{\mathbf{x}^0_n}\mathbb{E}_{\mathbf{x}^s_{1:n}}\langle M_\theta(s,\mathbf{x}^s_{1:n},\mathbf{x}^0_{1:n-1}), \nabla_{\mathbf{x}^s_{1:n}} \log p(\mathbf{x}^s_{1:n}|\mathbf{x}^0_{1:n})\rangle \\
&= \mathbb{E}_{\mathbf{x}^0_n}\mathbb{E}_{\mathbf{x}^s_{1:n}}\left[\left\|M_\theta(s,\mathbf{x}^s_{1:n},\mathbf{x}^0_{1:n-1}) - \nabla_{\mathbf{x}^s_{1:n}} \log p(\mathbf{x}^s_{1:n}|\mathbf{x}^0_{1:n})\right\|_2^2\right] + C
\end{aligned}
\tag{19}
$$

C is a constant that does not depend on the parameter θ. □

Corollary A.2. *Our target objective function, L_{score}, is defined as follows:*

$$
L_{score} = \mathbb{E}_s\mathbb{E}_{\boldsymbol{x}^0_{1:N}}\left[\sum_{n=1}^{N}\lambda(s) l_2^\star(n,s)\right], \tag{20}
$$

where

$$
l_2^\star(n,s) = \mathbb{E}_{\mathbf{x}^s_{1:n}}\left[\left\|M_\theta(s,\mathbf{x}^s_{1:n},\mathbf{x}^0_{1:n-1}) - \nabla_{\mathbf{x}^s_{1:n}} \log p(\mathbf{x}^s_{1:n}|\mathbf{x}^0_{1:n})\right\|_2^2\right]. \tag{21}
$$

Then, $L_2 = L_{score}$ is satisfied.

proof. Whereas one can use the law of total expectation, which means $E[X] = E[E[X|Y]]$ *if X,Y are on an identical probability space* to show the above formula, we calculate directly. At first, let us simplify the expectation of the inner part with a symbol $f(\mathbf{x}^0_{1:n})$ for our computational convenience, i.e., $f(\mathbf{x}^0_{1:n}) = \mathbb{E}_s\mathbb{E}_{\mathbf{x}^s_{1:n}}\left[\lambda(s)\left\|M_\theta(s,\mathbf{x}^s_{1:n},\mathbf{x}^0_{1:n-1}) - \nabla_{\mathbf{x}^s_{1:n}} \log p(\mathbf{x}^s_{1:n}|\mathbf{x}^0_{1:n})\right\|_2^2\right]$. Then we have the following definition:

$$
L_2 = \mathbb{E}_s\mathbb{E}_{\mathbf{x}^0_{1:N}}[l_2] = \mathbb{E}_{\mathbf{x}^0_{1:N}}\left[\sum_{n=1}^{N}\mathbb{E}_{\mathbf{x}^0_n}[f(\mathbf{x}^0_{1:n})]\right] = \sum_{n=1}^{N}\mathbb{E}_{\mathbf{x}^0_{1:N}}\mathbb{E}_{\mathbf{x}^0_n}[f(\mathbf{x}^0_{1:n})] \tag{22}
$$

At last, the expectation part can be further simplified as follows:

$$
\begin{aligned}
&\mathbb{E}_{\mathbf{x}^0_{1:N}}\mathbb{E}_{\mathbf{x}^0_n}[f(\mathbf{x}^0_{1:n})] \\
&= \int_{\mathbf{x}^0_{1:N}}\int_{\mathbf{x}^0_n} f(\mathbf{x}^0_{1:n})p(\mathbf{x}^0_n|\mathbf{x}^0_{1:n-1})d\mathbf{x}^0_n \cdot p(\mathbf{x}^0_{1:n-1})p(\mathbf{x}^0_{n:N}|\mathbf{x}^0_{1:n-1})d\mathbf{x}^0_{1:N} \\
&= \int_{\mathbf{x}^0_{1:N}}\int_{\mathbf{x}^0_n} f(\mathbf{x}^0_{1:n})p(\mathbf{x}^0_{1:n})d\mathbf{x}^0_n \cdot p(\mathbf{x}^0_{n:N}|\mathbf{x}^0_{1:n-1})d\mathbf{x}^0_{1:N} \\
&= \int_{\mathbf{x}^0_{n:N}}\left(\int_{\mathbf{x}^0_{1:n}} f(\mathbf{x}^0_{1:n})p(\mathbf{x}^0_{1:n})d\mathbf{x}^0_{1:n}\right) p(\mathbf{x}^0_{n:N}|\mathbf{x}^0_{1:n-1})d\mathbf{x}^0_{n:N} \\
&= \int_{\mathbf{x}^0_{1:n}} f(\mathbf{x}^0_{1:n})p(\mathbf{x}^0_{1:n})d\mathbf{x}^0_{1:n} \\
&= \int_{\mathbf{x}^0_{1:n}}\left(\int_{\mathbf{x}^0_{n+1:N}} p(\mathbf{x}^0_{n+1:N}|\mathbf{x}^0_{1:n})d\mathbf{x}^0_{n+1:N}\right) f(\mathbf{x}^0_{1:n})p(\mathbf{x}^0_{1:n})d\mathbf{x}^0_{1:n} \\
&= \int_{\mathbf{x}^0_{1:N}} f(\mathbf{x}^0_{1:n})p(\mathbf{x}^0_{1:N})d\mathbf{x}^0_{1:N} \\
&= \mathbb{E}_{\mathbf{x}^0_{1:N}}[f(\mathbf{x}^0_{1:n})]
\end{aligned}
\tag{23}
$$

Since $\sum_{n=1}^{N}\mathbb{E}_{\mathbf{x}^0_{1:N}}[f(\mathbf{x}^0_{1:n})] = \mathbb{E}_{\mathbf{x}^0_{1:N}}[\sum_{n=1}^{N} f(\mathbf{x}^0_{1:n})] = L_{score}$, we prove the corollary. □

B Existing Time-Series Diffusion Models

B.1 Diffusion Models for Time-series Forecasting and Imputation

TimeGrad (Rasul et al., 2021) is a diffusion-based method for time-series forecasting, and CSDI (Tashiro et al., 2021) is for time-series imputation.

In TimeGrad (Rasul et al., 2021), they used a diffusion model for forecasting future observations given past observations. On each sequential order $n \in \{2, ..., N\}$ and diffusion step $s \in \{1, ..., T\}$, they train a neural network $\boldsymbol{\epsilon}_\theta(\cdot, \mathbf{x}_{1:n-1}, s)$ with a time-dependent diffusion coefficient $\bar{\alpha}_s$ by minimizing the following objective function:

$$
\mathbb{E}_{\mathbf{x}^0_n,\epsilon,s}[\left\|\epsilon - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_s}\mathbf{x}^0_n + \sqrt{1-\bar{\alpha}_s}\boldsymbol{\epsilon}, \mathbf{x}_{1:n-1}, s)\right\|_2^2], \tag{24}
$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. The above formula assumes that we already know $\mathbf{x}_{1:n-1}$, and by using an RNN encoder, $\mathbf{x}_{1:n-1}$ can be encoded into $\mathbf{h}_{n-1}$. After training, the model forecasts future observations recursively. More precisely speaking, $\mathbf{x}_{1:n-1}$ is encoded into $\mathbf{h}_{n-1}$ and the next observation $\mathbf{x}_n$ is forecast from the previous condition $\mathbf{h}_{n-1}$.

CSDI (Tashiro et al., 2021) proposed a general diffusion framework which can be applied mainly to time-series imputation. CSDI reconstructs an entire sequence at once, not recursively. Let $\mathbf{x}^0 \in \mathbb{R}^{\dim(\mathbf{X})\times N}$ be an entire time-series sequence with N observations in a matrix form. They define $\mathbf{x}^0_{co}$ and $\mathbf{x}^0_{ta}$ as conditions and imputation targets which are derived from $\mathbf{x}^0$, respectively. They then train a neural network $\epsilon_\theta(\cdot, \mathbf{x}^0_{co}, s)$ with a corresponding diffusion coefficient $\bar{\alpha}_s$ and a diffusion step $s \in \{1, ..., T\}$ by minimizing the following objective function:

$$
\mathbb{E}_{\mathbf{x}^0,\epsilon,s}[\left\|\epsilon - \epsilon_\theta(s, \mathbf{x}^s_{ta}, \mathbf{x}^0_{co})\right\|_2^2], \tag{25}
$$

where $\mathbf{x}^s_{ta} = \sqrt{\bar{\alpha}_s}\mathbf{x}^0_{ta} + (1-\bar{\alpha}_s)\epsilon$. By training the network using the above loss, it generates missing elements from the partially filled matrix $\mathbf{x}^0_{co}$.

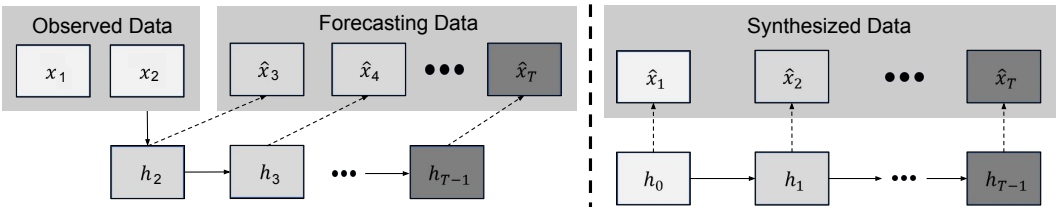

Figure 4: Graphical representation of TimeGrad (left) and TSGM (right). We adapt TimeGrad to our generation task but its results are not comparable even to other baselines' results (see Appendix C.1).

B.2 Difference between Existing and Our Works

Although they have earned state-of-the-art results for forecasting and imputation, we found that they are not suitable for our generative task due to the fundamental mismatch between their model designs and our task (cf. Table 6 and Fig. 4).

Table 6: Comparison among various recent GAN, diffusion, and SGM-based methods for time-series. $\mathbf{x}_t$ (resp. $\hat{\mathbf{x}}_t$) means a raw (resp. synthesized) observation at time t. For CSDI, $\mathbf{x}_{co}$ means a set of known values and $\mathbf{x}_{ta}$ means a set of target missing values — it is not necessary that $\mathbf{x}_{co}$ precedes $\mathbf{x}_{ta}$ in time in CSDI.

Method	Type	Task Description
TimeGrad	Diffusion	From $\mathbf{x}_{1:N-K}$, infer $\hat{\mathbf{x}}_{N-K+1:N}$.
CSDI	Diffusion	Given known values $\mathbf{x}_{co}$, infer missing values $\hat{\mathbf{x}}_{ta}$.
TimeGAN	GAN	Synthesize $\hat{\mathbf{x}}_{1:N}$ from scratch.
GT-GAN	GAN	Synthesize $\hat{\mathbf{x}}_{1:N}$ from scratch.
TSGM	SGM	Synthesize $\hat{\mathbf{x}}_{1:N}$ from scratch.

TimeGrad generates future observations given the hidden representation of past observations $\mathbf{h}_{n-1}$, i.e., a typical forecasting problem. Since our task is to synthesize from scratch, past known observations are not available. Thus, TimeGrad cannot be directly applied to our task.

In CSDI, there are no fixed temporal dependencies between $\mathbf{x}^0_{co}$ and $\mathbf{x}^0_{ta}$ since its task is to impute missing values, i.e., $\mathbf{x}^0_{ta}$, from known values, i.e., $\mathbf{x}^0_{co}$, in the matrix $\mathbf{x}^0$. It is not necessary that $\mathbf{x}^0_{co}$ precedes $\mathbf{x}^0_{ta}$ in time, according to the CSDI's method design. Our synthesis task can be considered as $\mathbf{x}^0_{co} = \emptyset$, which is the most extreme case of the CSDI's task. Therefore, it is not suitable to be used for our task.

To our knowledge, we are the first proposing an SGM-based time-series synthesis method. We propose to train a conditional score network by using the denoising score matching loss proposed by us, which is denoted as $L^{\mathcal{H}}_{score}$. Unlike other methods (Rasul et al., 2021; Tashiro et al., 2021) that resort to existing known proofs, we design our denoising score matching loss in Eq. equation 12 and prove its correctness. Meanwhile, TimeGrad and CSDI can be somehow modified for time-series synthesis but their generation quality is mediocre (see Appendix C).

C Experimental Results for Inapplicability of Existing Time-Series Diffusion Models to Our Work

In this section, we provide experimental results to show inapplicability of the existing time-series diffusion models, TimeGrad and CSDI, to the time-series generation task.

C.1 Adapting TimeGrad toward Generation Task

In this section, TimeGrad (Rasul et al., 2021) is modified for the generation task. We simply add an artificial zero vector $\mathbf{0}$ in front of the all time-series samples of Energy. Therefore, TimeGrad's task becomes given a zero vector, forecasting (or generating) all other remaining observations. For the stochastic nature of its forecasting process, it can somehow generate various next observations given

Table 7: Comparison between TSGM and modified TimeGrad in Energy for its regular time-series setting

Method	Disc.	Pred.
TSGM-VP	.221±.025	.257±.000
TSGM-subVP	.198±.025	.252±.000
Modified TimeGrad	.500±.000	.287±.003

Table 8: Comparison between TSGM and modified CSDI in Stock and Energy for its regular time-series setting

Method	Energy		AI4I	
	Disc.	Pred.	Disc.	Pred.
TSGM-VP	.221±.025	.257±.000	.147±.005	.217±.000
TSGM-subVP	.198±.025	.252±.000	.150±.010	.217±.000
Modified CSDI	.500±.000	.641±.000	.500±.000	.640±.000

the sample input $\mathbf{0}$. Table 7 shows the experimental comparison between modified TimeGrad and TSGM in Energy for its regular time-series setting. TSGM gives outstanding performance, compared to modified TimeGrad. When checked in Table 2, modified TimeGrad is even worse than some baselines. Therefore, unlike TSGM, TimeGrad is not appropriate for the generation task.

C.2 Adapting CSDI toward Generation Task

In this section, we apply CSDI to the time-series generation task by regarding all observations as missing values (i.e., $\mathbf{x}^0_{co} = \mathbf{0}$). However, as demonstrated in Table 8, CSDI fails to generate reliable time series samples in the Energy and AI4I datasets for its regular time series setting. Hence, we conclude that CSDI is unsuitable for the time-series generation task.

D Training Algorithm

Algorithm 1: Training algorithm

Input: $\mathbf{x}^0_{1:N}$; use_{alt} is a Boolean parameter to set whether to use the alternating training method; $iter_{pre}$ is the number of iterations for pre-training; $iter_{main}$ is the number of iterations for training.

for $iter \in \{1, ..., iter_{pre}\}$ **do**
  Train $Encoder$ and $Decoder$ by using L_{ed}
end
for $iter \in \{1, ..., iter_{main}\}$ **do**
  Train M_θ by using $L^{\mathcal{H}}_{score}$
  if use_{alt} **then**
    Train the $Encoder$ and $Decoder$ by using L_{ed}
  end
end
return $Encoder, Decoder, M_\theta$

E Datasets and Baselines

We use 4 datasets from various fields as follows. We summarize their data dimensions, the number of training samples, and their time-series lengths (window sizes) in Table 9.

- *Stock* (Yoon et al., 2019): The Google stock dataset was collected irregularly from 2004 to 2019. Each observation has (volume, high, low, opening, closing, adjusted closing prices), and these features are correlated.
- *Energy* (Candanedo et al., 2017): This dataset is from the UCI machine learning repository for predicting the energy use of appliances from highly correlated variables such as house temperature and humidity conditions.

- *Air* (De Vito et al., 2008): The UCI Air Quality dataset was collected from 2004 to 2005. Hourly averaged air quality records are gathered using gas sensor devices in an Italian city.
- *AI4I* (Matzka, 2020): AI4I means the UCI AI4I 2020 Predictive Maintenance dataset. This data reflects the industrial predictive maintenance scenario with correlated features including several physical quantities.

We use several types of generative methods for time-series as baselines. At first, we consider autoregressive generative methods: T-Forcing (teacher forcing) (Graves, 2013; Sutskever et al., 2011) and P-Forcing (professor forcing) (Goyal et al., 2016). Next, we use GAN-based methods: TimeGAN (Yoon et al., 2019), RCGAN (Esteban et al., 2017), C-RNN-GAN (Mogren, 2016), COT-GAN (Xu et al., 2020), GT-GAN (Jeon et al., 2022). We also test VAE-based methods into our baselines: TimeVAE (Desai et al., 2021). Finally, we treat flow-based methods. Among the array of flow-based models designed for time series generation, we have chosen to compare our TSGM against CTFP (Deng et al., 2020). This choice is informed by the fact that CTFP possesses the capability to handle both regular and irregular time series samples, aligning well with the nature of our task which involves generating both regular and irregular time series data.

Table 9: Characteristics of the datasets we use for our experiments

Dataset	Dimension	#Samples	Length
Stocks	6	3685	24
Energy	28	19735	
Air	13	9357	
AI4I	5	10000	

F Hyperparameters and its Search Space

Table 10 shows the best hyperparameters for our conditional score network M_θ on regular time-series, and we explain its neural network architecture in Appendix J.2. M_θ has various hyperparameters and for key hyperparameters, we set them as listed in Table 10. For other common hyperparameters with baselines, we reuse the default configurations of TimeGAN (Yoon et al., 2019) and VPSDE (Song et al., 2021) to conduct the regular time-series generation.

We give our search space for the hyperparameters of TSGM. $iter_{pre}$ is in {50000,100000}. The dimension of hidden features, d_{hidden}, ranges from 2 times to 5 times the dimension of input features. On regular time-series generation, we follow the default values in TimeGAN (Yoon et al., 2019) and VPSDE (Song et al., 2021). For irregular time-series tasks, we search the hidden dimension of decoder from 2 times to 4 times the dimension of input dimension, and follow GTGAN (Jeon et al., 2022) for other settings of NCDE-encoder and GRU-ODE-decoder. We give our best hyperparameters for irregular time-series on Table 11.

For baselines, we check their hyperparameters as follow:

- T-forcing (Graves, 2013): We control batch size among {256, 512, 1024}.
- P-forcing (Goyal et al., 2016): We control batch size among {256, 512, 1024}.
- TimeGAN (Yoon et al., 2019): The dimension of hidden features range from 2 times to 4 times the dimension of input features.
- RCGAN (Esteban et al., 2017): We control learning rate of generator's optimizer and discriminator's optimizer from {1e-4, 2e-4} and {1e-3, 5e-3}, respectively.
- C-RNN-GAN (Mogren, 2016): We control learning rate of generator's optimizer and discriminator's optimizer from {1e-4, 2e-4} and {3e-4, 4e-4}, respectively. We also use label smoothing which is stated in the paper.
- TimeVAE (Desai et al., 2021): We control its latent dimension among {5, 10, 20}.
- COT-GAN (Xu et al., 2020): We calculate score every 250 epoch during 1000 epochs and get the best experimental results.

- CTFP (Deng et al., 2020): The dimension of hidden features range from 2 times to 4 times the dimension of input features.
- GT-GAN (Jeon et al., 2022): For encoder-decoder pair, we test from exactly the same search space as TSGM. We calculate score every 5000 iteration during 40000 iterations and get the best score.

Especially for COT-GAN, since it is on video generation, modifying the architecture to one dimensional form was difficult. So, we augment our time-series data into two dimensional ones by stacking them. After generating two-dimensional data, we extract the first row of the synthesized one and calculate the score. We search every hyperparameter from {0.5, 1, 2} times of default value. Through the experiment, we acquire compatible result but lower than TimeGAN in several datasets.

We follow default values for miscellaneous settings which are not explained on the above. Additionally, to deal with irregular time-series, we search the hyperparameters of GRU-D, which substitutes for RNN or are added to the head of baselines. We test the hidden dimension of GRU-D from 2 times to 4 times the dimension of input features.

Table 10: The best hyperparameter setting for our method on regular time-series.

Dataset	$\dim(\mathbf{h})$	use_{alt}	$iter_{pre}$	$iter_{main}$
Stocks	24	True	50000	40000
Energy	56	False	100000	
Air	40	True	50000	
AI4I	24	True	50000	

Table 11: The best hyperparameter setting for our method on irregular time-series. D_{hidden} denotes the hidden dimension of GRU-ODE-decoder.

Dataset	D_{hidden}	$\dim(\mathbf{h})$	use_{alt}	$iter_{pre}$	$iter_{main}$
Stocks	48	24	True	50000	40000
Energy	112	56	False		
Air	40	40	True		
AI4I	48	24	True		

G Miscellaneous Experimental Environments

We give detailed experimental environments. The following software and hardware environments were used for all experiments: Ubuntu 18.04 LTS, Python 3.9.12, CUDA 9.1, NVIDIA Driver 470.141, i9 CPU, and GeForce RTX 2080 Ti.

In the experiments, we report only the VP and subVP-based TSGM and exclude the VE-based one for its lower performance. For baselines, we reuse their released source codes in their official repositories and rely on their designed training and model selection procedures. For our method, we select the best model for every 5000 iterations. For this, we synthesize samples and calculate the mean and standard deviation scores of the discriminative and predictive scores.

H Empirical Space and Time Complexity Analyses

We report the memory usage during training in Table 12 and the wall-clock time for generating 1,000 time-series samples in Table 13. We compare TSGM to TimeGAN (Yoon et al., 2019) and GTGAN (Jeon et al., 2022). Our method is relatively slower than TimeGAN and GTGAN, which is a fundamental drawback of all SGMs. For example, the original score-based model (Song et al., 2021) requires 3,214 seconds for sampling 1,000 CIFAR-10 images while StyleGAN (Karras et al., 2019) needs 0.4 seconds. However, we also emphasize that this problem can be relieved by using the techniques suggested in (Xiao et al., 2022; Jolicoeur-Martineau et al., 2021) as we mentioned in the conclusion section.

Table 12: The memory usage for training

Method	Stock	Energy
TimeGAN	1.1 (GB)	1.6 (GB)
GTGAN	2.3 (GB)	2.3 (GB)
TSGM	3.8 (GB)	3.9 (GB)

Table 13: The sampling time of TSGM, TimeGAN and GTGAN for generating 1,000 samples on each dataset. The original score-based model (Song et al., 2021) requires 3,214 seconds for sampling 1000 CIFAR-10 images while StyleGAN (Karras et al., 2019) needs 0.4 seconds, which is similar to the case between TSGM and TimeGAN.

Method	Stocks	Energy
TimeGAN	0.43 (s)	0.47 (s)
GTGAN	0.43 (s)	0.47 (s)
TSGM	3318.99 (s)	1620.84 (s)

I Encoder and Decoder for Irregular Time-series

To process irregular time-series, one can use continuous-time methods for constructing the encoder and the decoder. In our case, we use neural controlled differential equations (NCDEs) for designing the encoder and GRU-ODEs for designing the decoder, respectively (Kidger et al., 2020; Brouwer et al., 2019). Our encoder based on NCDEs can be defined as follows:

$$\mathbf{h}(t_n) = \mathbf{h}(t_{n-1}) + \int_{t_{n-1}}^{t_n} f(t, \mathbf{h}(t); \theta_f) \frac{dX(t)}{dt} dt, \tag{26}$$

where $X(t)$ is an interpolated continuous path from $\mathbf{x}_{1:N}$ — NCDEs typically use the natural cubic spline algorithm to define $X(t)$, which is twice differentiable and therefore, there is not any problem to be used for forward inference and backward training. In other words, NCDEs evolve the hidden state $\mathbf{h}(t)$ by solving the above Riemann-Stieltjes integral.

For the decoder, one can use the following GRU-ODE-based definition:

$$\overline{\mathbf{d}}(t_n) = \mathbf{d}(t_{n-1}) + \int_{t_{n-1}}^{t_n} g(t, \mathbf{d}(t); \theta_g) dt, \qquad \mathbf{d}(t_n) = \text{GRU}(\mathbf{h}(t_n), \overline{\mathbf{d}}(t_n)), \qquad \hat{\mathbf{x}}_n = FC(\mathbf{d}(t_n)), \tag{27}$$

where FC denotes a fully-connected layer-based output layer. The intermediate hidden representation $\overline{\mathbf{d}}(t_n)$ is jumped into the hidden representation $\mathbf{d}(t_n)$ by the GRU-based jump layer. At the end, there is an output layer.

For our irregular time-series experiments, i.e, dropping 30%, 50%, and 70% of observations from regular time-series, we use the above encoder and decoder definitions and have good results.

J Neural Network Architecture

J.1 Architectural Details of NCDEs and GRU-ODEs

As mentioned in Appendix I, we take the following architecture for functions f, g of (26) and (27) in Table 14.

J.2 Conditional Score Network

Unlike other generation tasks, e.g., image generation (Song et al., 2021) and tabular data synthesis (Kim et al., 2022), where each sample is independent, time-series observations are dependent to their past observations. Therefore, the score network for time-series generation must be designed to

Table 14: Architecture of functions f(upper) and g(lower). Each layer of encoder and gate of decoder takes $(\sigma \circ \text{Linear})$ form where σ denotes activation function. We describe which activation and Linear function are used.

Layer	Activation function	Linear
1	ReLU	dim(**x**)$\rightarrow$ 4 dim(**x**)
2	ReLU	4 dim(**x**)$\rightarrow$ 4 dim(**x**)
3	ReLU	4 dim(**x**)$\rightarrow$ 4 dim(**x**)
4	Tanh	4 dim(**x**)$\rightarrow$ dim(**x**)

Layer	Gate	Activation function	Linear
1	r_t	ReLU	dim(**h**)$\rightarrow$ dim(**h**)
	z_t	ReLU	
	u_t	Tanh	

Table 15: Experimental results in terms of the discriminative and predictive scores. The best scores are in boldface. The left and right ones denote experimental results on irregular time-series with 50% and 70% missing rates, respectively.

	Method	Stocks	Energy	Air	AI4I		Method	Stocks	Energy	Air	AI4I
Disc. score	TSGM-VP	.051±.014	.398±.003	.272±.012	.156±.106	Disc. score	TSGM-VP	.065±.010	.482±.003	.337±.025	.327±.104
	TSGM-subVP	**.031±.012**	.421±.008	**.213±.025**	**.137±.102**		TSGM-subVP	**.035±.009**	**.213±.025**	**.329±.027**	**.235±.123**
	T-Forcing-D	.407±.034	.376±.046	.499±.001	.473±.045		T-Forcing-D	.404±.068	.336±.032	.499±.001	.493±.010
	P-Forcing-D	.500±.000	.500±.000	.494±.012	.437±.079		P-Forcing-D	.449±.150	.494±.011	.498±.002	.440±.125
	TimeGAN-D	.477±.021	.473±.015	.500±.001	.500±.000		TimeGAN-D	.485±.022	.500±.000	.500±.000	.500±.000
	RCGAN-D	.500±.000	.500±.000	.500±.000	.500±.000		RCGAN-D	.500±.000	.500±.000	.500±.000	.500±.000
	C-RNN-GAN-D	.500±.000	.500±.000	.500±.000	.450±.150		C-RNN-GAN-D	.500±.000	.500±.000	.500±.000	.500±.000
	TimeVAE-D	.411±.110	.436±.088	.423±.153	.389±.113		TimeVAE-D	.444±.148	.498±.003	.426±.148	.371±.092
	COT-GAN-D	.499±.001	.500±.000	.500±.000	.500±.000		COT-GAN-D	.498±.001	.500±.000	.500±.000	.500±.000
	CTFP	.499±.000	.500±.000	.500±.000	.499±.001		CTFP	.500±.000	.500±.000	.500±.000	.499±.000
	GT-GAN	.265±.073	**.317±.010**	.434±.035	.276±.033		GT-GAN	.230±.053	.325±.047	.444±.019	.362±.043
Pred. score	TSGM-VP	.011±.000	.051±.001	**.041±.001**	**.060±.001**	Pred. score	TSGM-VP	**.011±.000**	.053±.001	.043±.000	.092±.024
	TSGM-subVP	**.011±.000**	**.051±.001**	.042±.002	.065±.013		TSGM-subVP	.012±.000	**.042±.002**	**.042±.001**	**.097±.020**
	T-Forcing-D	.038±.003	.090±.000	.121±.003	.143±.005		T-Forcing-D	.031±.002	.091±.000	.116±.003	.144±.004
	P-Forcing-D	.089±.010	.198±.005	.101±.003	.116±.007		P-Forcing-D	.107±.009	.193±.006	.107±.002	.125±.007
	TimeGAN-D	.254±.047	.339±.029	.325±.005	.251±.010		TimeGAN-D	.228±.000	.443±.000	.425±.008	.323±.011
	RCGAN-D	.333±.044	.250±.010	.335±.023	.276±.066		RCGAN-D	.441±.045	.349±.027	.359±.008	.346±.029
	C-RNN-GAN-D	.273±.000	.438±.000	.289±.033	.373±.037		C-RNN-GAN-D	.281±.019	.436±.000	.306±.040	.262±.053
	TimeVAE-D	.195±.012	.143±.007	.103±.002	.144±.004		TimeVAE-D	.199±.009	.134±.004	.108±.004	.142±.008
	COT-GAN-D	.246±.000	.475±.000	.557±.000	.449±.000		COT-GAN-D	.278±.000	.456±.000	.556±.000	.435±.000
	CTFP	.084±.005	.469±.008	.476±.235	.412±.024		CTFP	.084±.005	.469±.008	.476±.235	.412±.024
	GT-GAN	.018±.002	.064±.001	.061±.003	.113±.024		GT-GAN	.020±.005	.076±.001	.059±.004	.124±.003
	Original	.011±.002	.045±.001	.044±.006	.059±.001		Original	.011±.002	.045±.001	.044±.006	.059±.001

learn the conditional log-likelihood given past generated observations, which is more complicated than that in image generation.

In order to learn the conditional log-likelihood, we modify the popular U-net (Ronneberger et al., 2015) architecture for our purposes. Since U-net has achieved various excellent results for other generative tasks (Song & Ermon, 2019; Song et al., 2021), we modify its 2-dimensional convolution layers to 1-dimensional ones for handling time-series observations. The modified U-net, denoted M_θ, is trained to learn our conditional score function (cf. Eq. equation 12). More details on training and sampling with M_θ are in Sec. 3.4.

K Additional Experimental Results

We give additional experimental results for irregular time-series generation with 50% and 70% missing rates in Table 15.

L Additional Visualizations

In this section, we provide additional visualization results in each dataset. Figure 5 illustrates the density function of each feature estimated by KDE in original and generated data. Figure 6 shows original and generated data points projected onto a latent space using t-SNE (van der Maaten & Hinton, 2008)

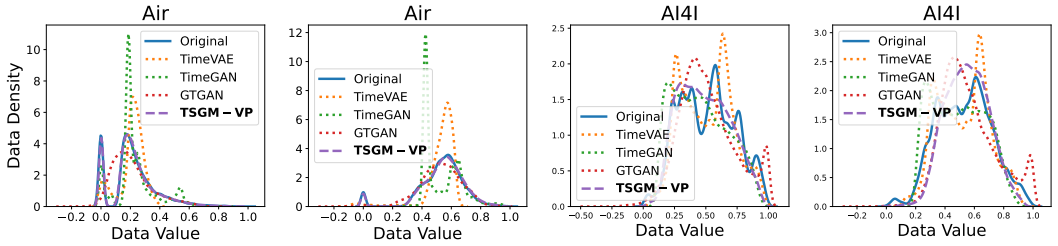

Figure 5: Additional KDE plots for each feature in Air and AI4I datasets.

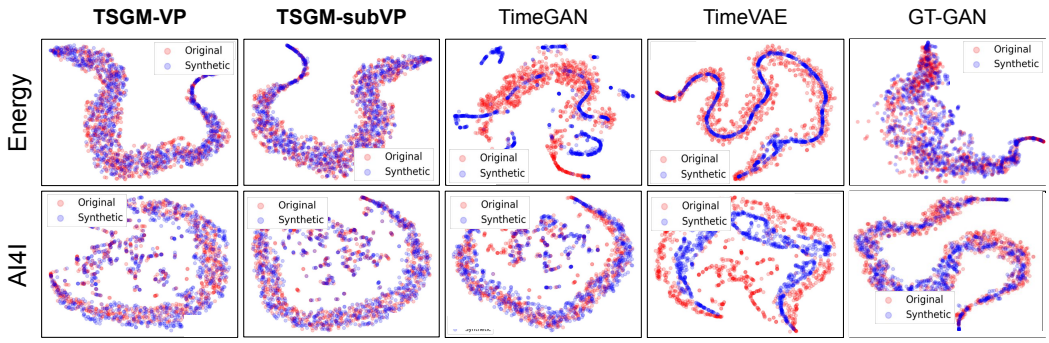

Figure 6: Additional t-SNE plots in Energy and AI4I datasets.

M Efficacy of Our Recursive Generation

In this section, we investigate the efficacy of our proposed recursive design. We compare TSGM to an method using *one-shot generation*. we call *one-shot generation* when a generation method generates all time-series observations at once, not recursively. In other words, $D \times N$ matrices, where D means the number of features and N means the sequence length, are synthesized at once. CSDI (Tashiro et al., 2021) is one of the most famous one-shot imputation model for time-series.

We convert our TSGM for the one-shot generation by removing the RNN-based encoder. In Table 16, TSGM-oneshot shows poor generation quality in Stock and Energy. TSGM-oneshot achieves comparable predictive scores but its discriminative score gets worse a lot. From these results, we can support the efficacy of our recursive structures, compared to one-shot generation. One can also check the one-shot generation result by CSDI in Appendix C.2.

Table 16: Comparison between TSGM and one-shot generations. We give representative results. For other datasets, the results are similar or worse than the table.

Method	Stock		Energy	
	Disc.	Pred.	Disc.	Pred.
TSGM-VP	.022±.005	.037±.000	.221±.025	.257±.000
TSGM-subVP	.021±.008	.037±.000	.198±.025	.252±.000
TSGM-oneshot	.029±.018	.037±.000	.494±.001	.258±.000